FACULTÉ DES SCIENCES DHAR EL MAHRAZ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH

Information Retrieval models

El Habib NFAOUI (elhabib.nfaoui@usmba.ac.ma)
LIIAN Laboratory, Faculty of Sciences Dhar Al Mahraz, Fez
Sidi Mohamed Ben Abdellah University, Fez

2018-2019

Outline

1. Basic Concepts of Information Retrieval
2. Information retrieval models
3. Vector space models
4. TF-IDF (Term Frequency-Inverse Document Frequency)
5. Doc2Vec (word2vec)
6. Latent Semantic Analysis (or Indexing) (LSA or LSI)
7. Information retrieval system evaluation

Introduction

- Relevance can be measured with respect to any of the following criteria
  - Query based
    - Similarity between query keywords and document is calculated
    - Can be enhanced through additional information such as popularity (Google)
  - Document
    - Measure of how useful a given document is in a given situation
  - User Based
    - Often associated with personalization, profile for a particular user is created
    - Similarity between a profile and document is calculated, No query is necessary
  - Role/Task Based
    - Similar to User Based Relevance
    - Profile is based on a particular role or task, instead of an individual

1. Basic Concepts of Information Retrieval

Information retrieval (IR) is the study of helping users to find information that matches their information needs. Technically, IR studies the acquisition, organization, storage, retrieval, and distribution of information. Historically, IR is about document retrieval, emphasizing document as the basic unit.

Fig. 1 gives a general architecture of an IR system.

In Fig. 1, the user with information need issues a query (**user query**) to the **retrieval system** through the **query operations** module. The retrieval module uses the **document index** to retrieve those documents that contain some query terms (such documents are likely to be relevant to the query), compute relevance scores for them, and then rank the retrieved documents according to the scores. The ranked documents are then presented to the user. The **document collection** is also called the **text database**, which is indexed by the **indexer** for efficient retrieval.

1. Basic Concepts of Information Retrieval

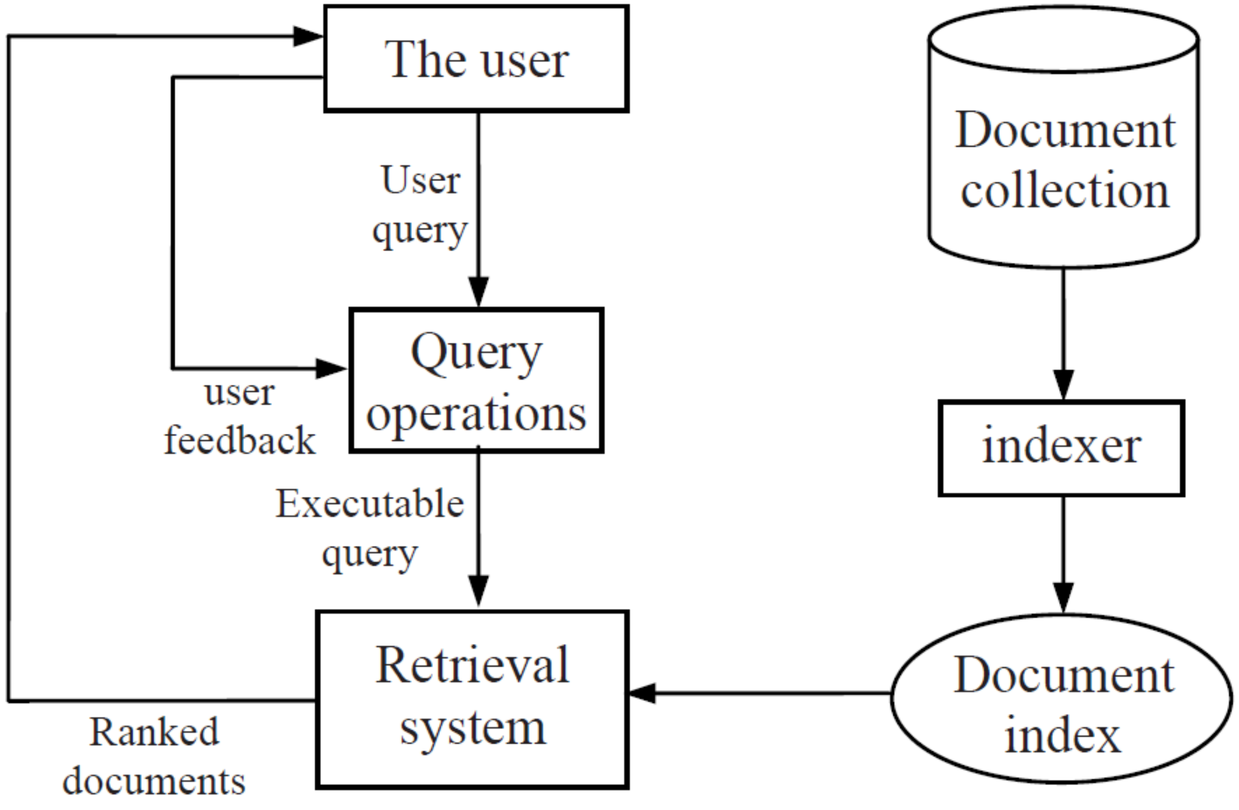


Figure 1. A general IR system architecture (Bing Liu, 2011)

1. Basic Concepts of Information Retrieval

- A **user query** represents the user's information needs, which is in one of the following forms:
  1. **Keyword queries**: The user expresses his/her information needs with a list of (at least one) keywords (or **terms**) aiming to find documents that contain some (at least one) or all the query terms. The terms in the list are assumed to be connected with a "soft" version of the logical AND. For example, if one is interested in finding information about Web mining, one may issue the query 'Web mining' to an IR or search engine system. 'Web mining' is retreated as 'Web AND mining'. The retrieval system then finds those likely relevant documents and ranks them suitably to present to the user. Note that a retrieved document does not have to contain all the terms in the query. In some IR systems, the ordering of the words is also significant and will affect the retrieval results.

  2. **Boolean queries**: The user can use Boolean operators, AND, OR, and NOT to construct complex queries. Thus, such queries consist of terms and Boolean operators.

  3. **Phrase queries**: Such a query consists of a sequence of words that makes up a phrase. Each returned document must contain at least one instance of the phrase. In a search engine, a phrase query is normally enclosed with double quotes. For example, one can issue the following phrase query (including the double quotes), "Information retrieval techniques and models" to find documents that contain the exact phrase.

1. Basic Concepts of Information Retrieval

4. **Proximity queries:** The proximity query is a relaxed version of the phrase query and can be a combination of terms and phrases. Proximity queries seek the query terms within close proximity to each other. The closeness is used as a factor in ranking the returned documents or pages. For example, a document that contains all query terms close together is considered more relevant than a page in which the query terms are far apart. Some systems allow the user to specify the maximum allowed distance between the query terms. Most search engines consider both term proximity and term ordering in retrieval.
5. **Full document queries**: When the query is a full document, the user wants to find other documents that are similar to the query document. Some search engines (e.g., Google) allow the user to issue such a query by providing the URL of a query page.
6. **Natural language questions:** This is the most complex case, and also the ideal case. The user expresses his/her information need as a natural language question. The system then finds the answer. However, such queries are still hard to handle due to the difficulty of natural language understanding. Nevertheless, this is an active research area, called **question answering**.

- The query operations module can range from very simple to very complex. In the simplest case, it does nothing but just pass the query to the retrieval engine after some simple pre-processing, e.g., removal of stopwords (words that occur very frequently in text but have little meaning, e.g., “the”, “a”, “in”, etc). In more complex cases, it needs to transform natural language queries into executable queries. It may also accept user feedback and use it to **expand and refine** the original queries. This is usually called **relevance feedback**.

1. Basic Concepts of Information Retrieval

- The **indexer** is the module that indexes the original raw documents in some data structures to enable efficient retrieval. The result is the **document index**. For example, the **inverted index** which is a particular type of indexing scheme, is used in search engines and most IR systems. An inverted index is easy to build and very efficient to search.

- The **retrieval system** computes a relevance score for each indexed document to the query. According to their relevance scores, the documents are ranked and presented to the user. Note that it usually does not compare the user query with every document in the collection, which is too inefficient. Instead, only a small subset of the documents that contains at least one query term is first found from the index and relevance scores with the user query are then computed only for this subset of documents.

2. Information retrieval models

- The information retrieval methods are needed to find the most relevant documents to a given query. An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined. There are four main IR models: Boolean model, **vector space model**, language model and probabilistic model. The most commonly used models in IR systems and on the Web are the first three models. In this presentation, we discuss the **vector space models** and how to implement them.

3. Vector space models

Formally, given a vocabulary of V words, each web page d_i (or document) in a collection of N pages, can be thought of as a vector of words,

$d_i = w_{1i}, \ldots, w_{|V|i}$, where each word j belonging to the document i is represented by w_{ij}, which can be either a number (weight) or a vector depending on the chosen algorithm:

- Term frequency-inverse document frequency (TF-IDF), w_{ij}, is a real number
- **Latent Semantic Analysis (LSA)**, w_{ij}, is a real number (representation independent of the document i)
- Doc2Vec (or word2vec), w_{ij}, is a vector of real numbers (representation independent of the document i)

Slides 3, 5.2, 5.3, 5.7 and 6 borrow heavily from Isoni Andrea's Book.

3. Vector space models

Since the query can also be represented by a vector of words, $q = w_{1q}, \ldots w_{|V|q}$, the web pages most similar to the vector q are found by calculating a similarity measure between the query vector and each document. The most used similarity measure is called cosine similarity, for any document i given by:

$$cosine(d_i\ , q) = \frac{d_i \cdot q}{|d_i||q|} = \frac{\sum_{j=1}^{|V|} w_{ij} w_{jq}}{\sqrt{\sum_{j=1}^{|V|} w_{ij}^2} \sqrt{\sum_{j=1}^{|V|} w_{qj}^2}}$$

Note that there are other measures used in literature (**Okapi** and **pivoted normalization weighting**). It has been shown that **Okapi** variations are more effective than cosine for **short query** retrieval.

The following sub-sections will give some details about the three methods.

4. TF-IDF (Term Frequency-Inverse Document Frequency)

This method calculates w_{ij}, taking into account the fact that a word that appears many times and in a large number of pages is likely to be less important than a word that occurs many times but only in a subset of documents. It is given by the multiplication of two factors:

$w_{ij} = tf_{ij} \, x \, idf_j$ where:

- $tf_{ij} = \dfrac{f_{ij}}{max f_{ij}, \ldots, f_{|V|}}$ is the normalized frequency of the word j in the document i
- $idf_j = \log \dfrac{N}{df_j}$ is the inverse document frequency and df_j is the number of web pages that contain the word j

Postscript: tf-idf weighting has many variants.

4. TF-IDF (Term Frequency-Inverse Document Frequency)

In other words, TF-IDF assigns a weight to term t in document d as follows:

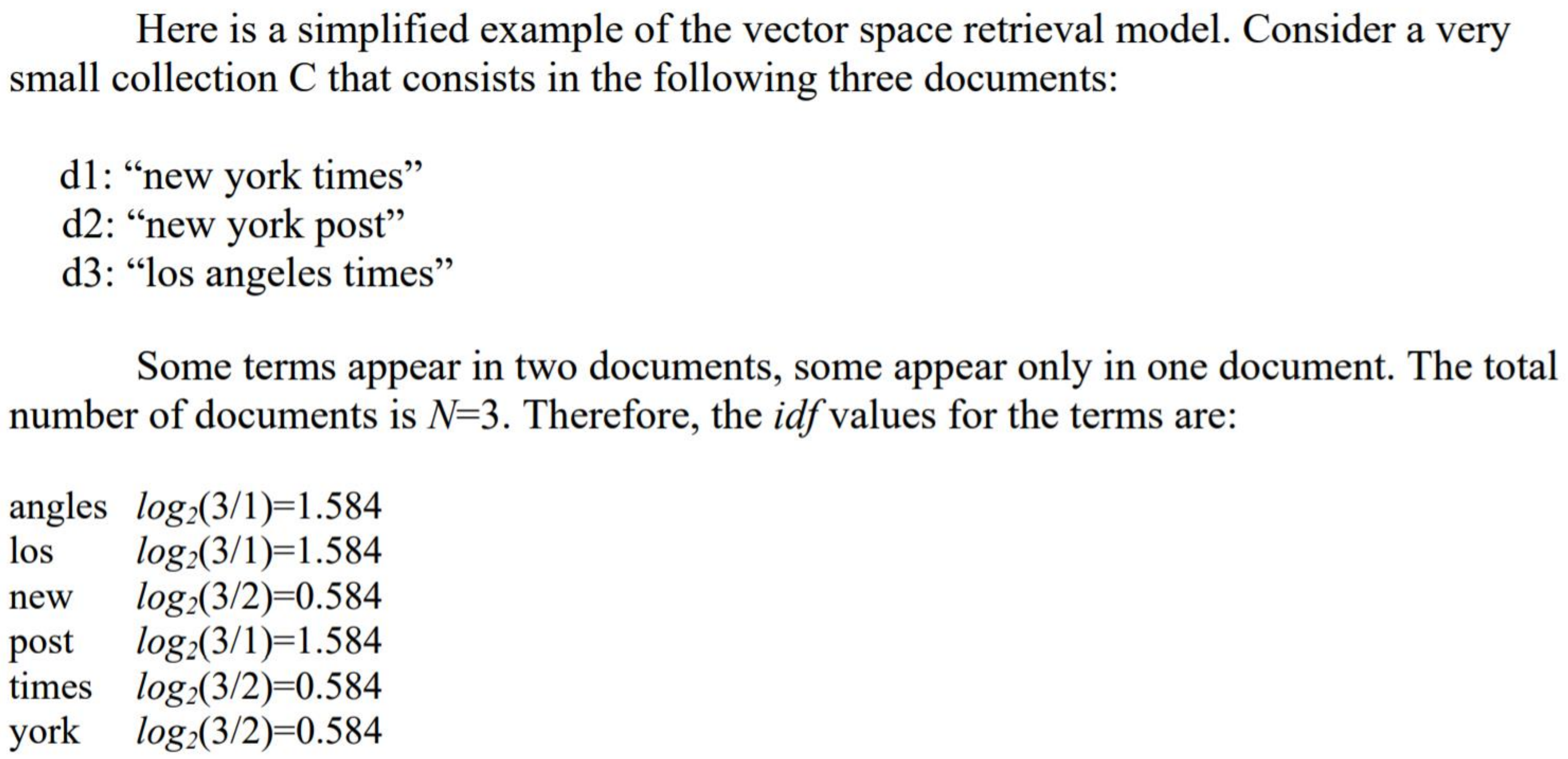
- If term t occurs many times in a few documents, it will be the highest.
- If term t occurs a small number of times in a document, it will be lower.
- If term t occurs in all documents, it will be the lowest.
- If term t occurs in no documents, it will be 0.

Queries:

A query **q** is represented in exactly the same way as a document in the document collection. The term weight $\mathbf{w_{jq}}$ of each term $\mathbf{t_j}$ in **q** can also be computed in the same way as in a normal document, or slightly differently. For example, Salton and Buckley (Salton and Buckley, 1988) suggested the following:

$$w_{jq} = \left(0.5 + \frac{0.5 f_{jq}}{\max\{f_{1q}, f_{2q}, \ldots, f_{|V|q}\}}\right) \times \log \frac{N}{df_j}$$

4.1 Computing TF-IDF: An Example



Here is a simplified example of the vector space retrieval model. Consider a very small collection C that consists in the following three documents:

d1: “new york times”
d2: “new york post”
d3: “los angeles times”

Some terms appear in two documents, some appear only in one document. The total number of documents is N=3. Therefore, the *idf* values for the terms are:

angles	$log_2(3/1)=1.584$
los	$log_2(3/1)=1.584$
new	$log_2(3/2)=0.584$
post	$log_2(3/1)=1.584$
times	$log_2(3/2)=0.584$
york	$log_2(3/2)=0.584$

4.1 Example (continued)

For all the documents, we calculate the *tf* scores for all the terms in C. We assume the words in the vectors are ordered alphabetically.

	angeles	los	new	post	times	york
d1	0	0	1	0	1	1
d2	0	0	1	1	0	1
d3	1	1	0	0	1	0

Now we multiply the *tf* scores by the *idf* values of each term, obtaining the following matrix of documents-by-terms: (All the terms appeared only once in each document in our small collection, so the maximum value for normalization is 1.)

	angeles	los	new	post	times	york
d1	0	0	0.584	0	0.584	0.584
d2	0	0	0.584	1.584	0	0.584
d3	1.584	1.584	0	0	0.584	0

4.1 Example (continued)

Given the following query: "new new times", we calculate the *tf-idf* vector for the query, and compute the score of each document in C relative to this query, using the cosine similarity measure. When computing the *tf-idf* values for the query terms we divide the frequency by the maximum frequency (2) and multiply with the *idf* values.

q	0	0	(2/2)*0.584=0.584	0	(1/2)*0.584=0.292	0

4.1 Example (continued)

We calculate the length of each document and of the query:

Length of d1 = sqrt(0.584^2+0.584^2+0.584^2)=1.011
Length of d2 = sqrt(0.584^2+1.584^2+0.584^2)=1.786
Length of d3 = sqrt(1.584^2+1.584^2+0.584^2)=2.316
Length of q = sqrt(0.584^2+0.292^2)=0.652

Then the similarity values are:

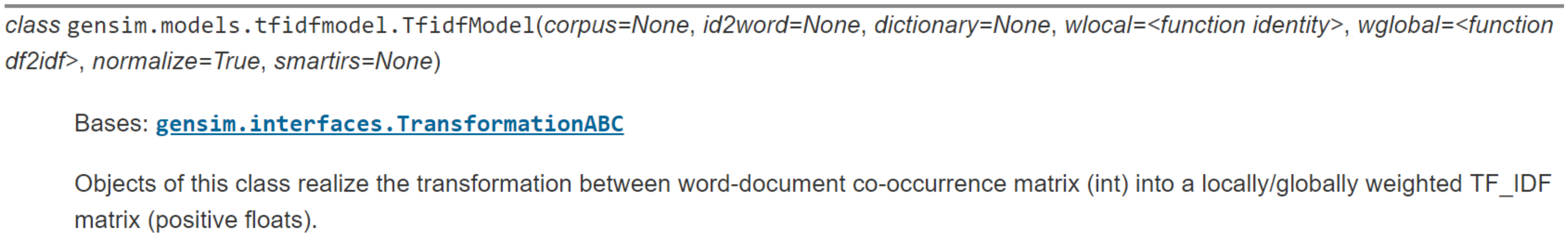
cosSim(d1,q) = (0*0+0*0+0.584*0.584+0*0+0.584*0.292+0.584*0) / (1.011*0.652) = 0.776
cosSim(d2,q) = (0*0+0*0+0.584*0.584+1.584*0+0*0.292+0.584*0) / (1.786*0.652) = 0.292
cosSim(d3,q) = (1.584*0+1.584*0+0*0.584+0*0+0.584*0.292+0*0) / (2.316*0.652) = 0.112

According to the similarity values, the final order in which the documents are presented as result to the query will be: d1, d2, d3.

4.2 Implementation: gensim Python Library

models.tfidfmodel – TF-IDF model

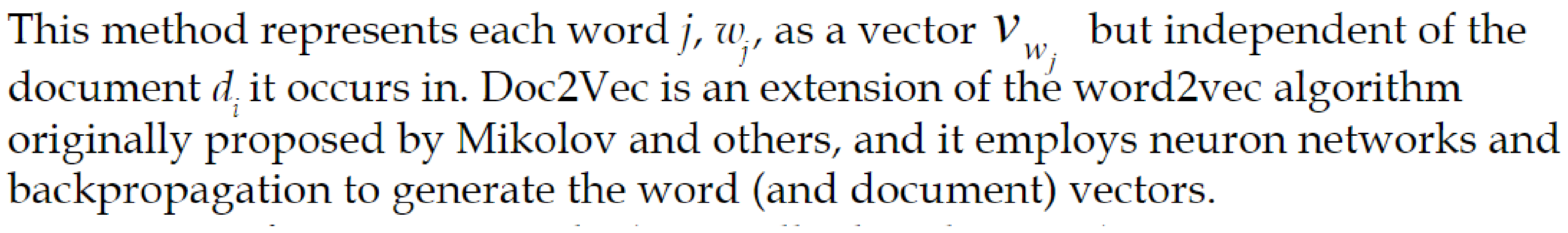
https://radimrehurek.com/gensim/models/tfidfmodel.html


class `gensim.models.tfidfmodel.TfidfModel`(*corpus=None*, *id2word=None*, *dictionary=None*, *wlocal=<function identity>*, *wglobal=<function df2idf>*, *normalize=True*, *smartirs=None*)

Bases: `gensim.interfaces.TransformationABC`

Objects of this class realize the transformation between word-document co-occurrence matrix (int) into a locally/globally weighted TF_IDF matrix (positive floats).

5. Doc2Vec (word2vec)


This method represents each word j, w_j, as a vector v_{w_j} but independent of the document d_i it occurs in. Doc2Vec is an extension of the word2vec algorithm originally proposed by Mikolov and others, and it employs neuron networks and backpropagation to generate the word (and document) vectors.

Word2vec takes a large amount of text data or text corpus as input and generates a set of vectors from the given text.

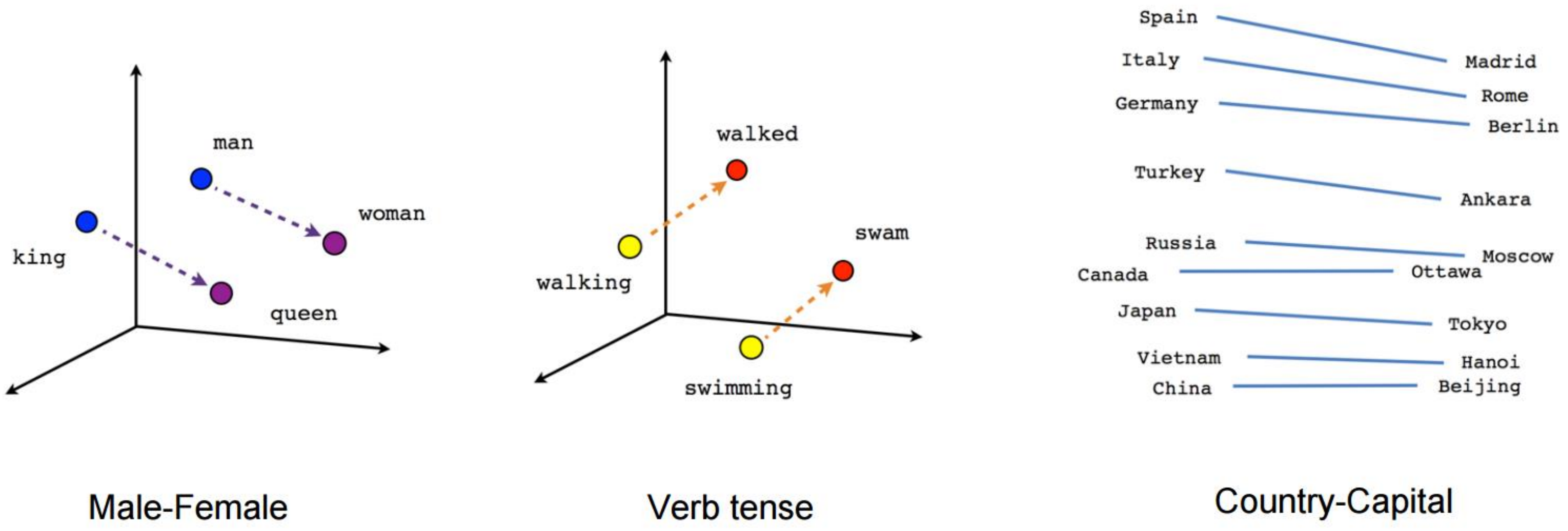
In other words, we can say that it generates high-dimensional vector space. This vector space has several hundred dimensions.

In the vector space, **each unique word in the corpus is assigned a corresponding vector**. Thus, the vector space is just a vector representation of all words present in the large text corpus.

In NLP, word2vec is one of the models that generates **word embedding**. It is a powerful, unsupervised word embedding technique.

5.1 word2vec learns semantic relationships between words

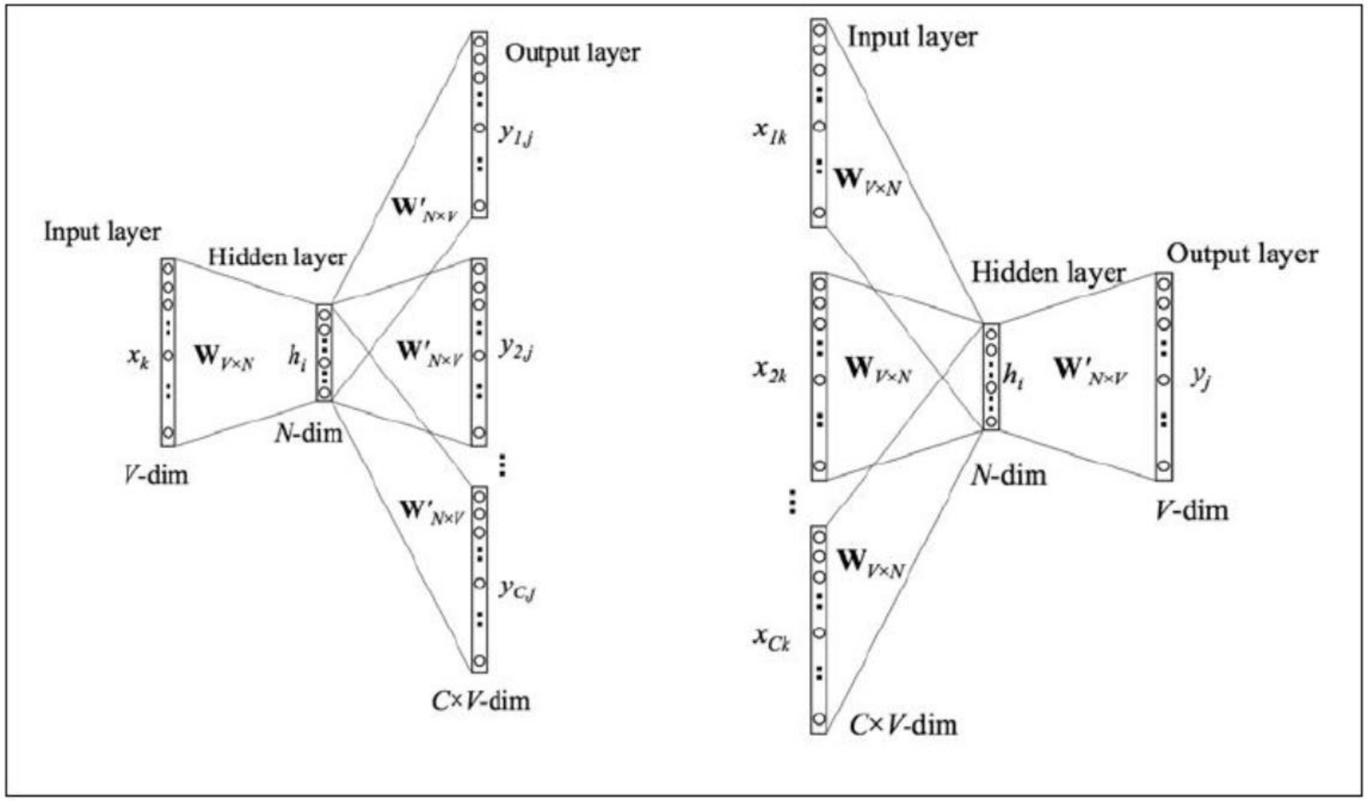
- We can visualize the learned vectors by projecting them down to 2 dimensions space using for instance the t-SNE dimensionality reduction technique. When we inspect these visualizations it becomes apparent that the vectors capture some general, and in fact quite useful, semantic information about words and their relationships to one another. It was very interesting when we first discovered that certain directions in the induced vector space specialize towards certain semantic relationships, e.g. *male-female*, *verb tense* and even *country-capital* relationships between words, as illustrated in the figure below.



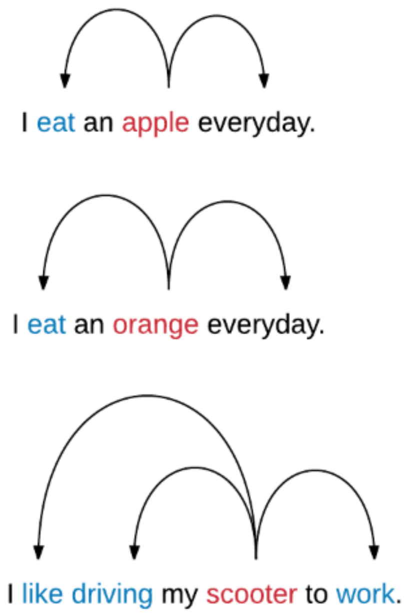
5.2 word2vec : continuous bag of words and skip-gram architectures

Each word *j* in the vocabulary *V* is represented by a vector of length |*V*|, with binary entries $x_j=(x_{1j}, \ldots, x_{Vj})$, where only $x_{jj}=1$; otherwise, *0*. The word2vec method trains a single (hidden) layer of *N* neurons (weights), choosing between two different network architectures (shown in the following figure). Note that both architectures have only one layer of *N* neurons or weights, *h*. This means that the method has to be considered *shallow* learning and not deep, which typically refers to networks with many hidden layers. The **Continuous Bag Of Words** (**CBOW**) method (displayed to the right in the following figure) trains the model using a set of *C* words as an input called *context* trying to predict the word (target) that occurs adjacent to the input text. The reverse approach is called **Skip-gram**, in which the input is the target word and the network is trained to predict the context set (displayed to the left in the following figure). Note that *C* is called the window parameter and it sets how far from the target word the context words are selected:

5.2 word2vec : continuous bag of words and skip-gram architectures



Skip-gram (left) and CBOW (right) architectures of the word2vec algorithm; figures taken from word2vec Parameter Learning Explained by X Rong (2015)



Target word and context words

5.2 word2vec : continuous bag of words and skip-gram architectures

In both cases, the matrix W transforms the input vectors into the hidden layer and W' transforms from the hidden layer to the output layer y, where the target (or context) is evaluated. In the training phase, the error from the true target (or context) is computed and used to calculate a stochastic gradient descent to update both the matrices W and W'. We will give a more mathematical description of the CBOW method in the following section. Note that the Skip-gram equations are similar and we will refer to the Rong (2015) paper for further details.

5.3 Mathematical description of the CBOW model

Starting from the input layer, the hidden layer h can be obtained by computing,

$h = \frac{1}{C} W \cdot (x_1 + .. + x_C) = \frac{1}{C}(v_{w_1} + .. + v_{w_C}) = v_C$, where v_{w_i} is a vector of length N that represents the word w_i on the hidden layer and $w_c \to v_c$ is the average of the C context vectors v_{w_i}. Choosing a target word w_j, the score at the output layer u_j is obtained by multiplying the vector v'_{w_j} (the j-th column of W') by h:

$$u_j = v'_{w_j} \cdot h$$

5.3 Mathematical description of the CBOW model

This is not the final value on the output layer y_j because we want to evaluate the posterior conditional probability to have the target word w_j given the context C expressed by the **softmax** formula:

$$y_j = p\left(w_j \mid w_{1,}., w_C \right) = \frac{e^{u_j}}{\sum_{i=1}^{|V|} e^{u_i}} = \frac{e^{v_j'^T v_C}}{\sum_{i=1}^{|V|} e^{v_i'^T v_C}}$$

Now the training objective is to maximize this probability

for all the words in the vocabulary, which is equivalent to

$$\max p\left(w_j \mid w_{1,}., w_C\right) \rightarrow E = -\max_j \log p\left(w_j \mid w_1,., w_C\right) = -v'^T_{w_{j^M}} \cdot h + \log \sum_{i=1}^{|V|} e^{v'_{w_i} h},$$

where $\max_j \left(v'^T_{w_j} \cdot h \right) = v'^T_{w_{j^M}} \cdot h$ and the index j^M represents the vector of W' in which the product is maximum, that is, the most probable target word.

5.3 Mathematical description of the CBOW model

The stochastic gradient descent equations are then obtained by calculating the derivatives of E with respect to the entries of W (w_{ij}) and W' (w'_{ij}). The final equations for each output target word w_j are:

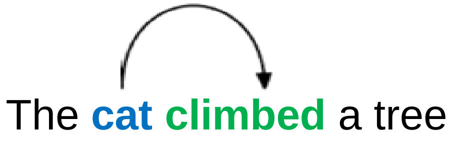
$$v'^{new}_{w_j} = v'^{old}_{w_j} - \alpha \frac{\partial E}{\partial u_j} \forall j \in 1, \dots |V|$$

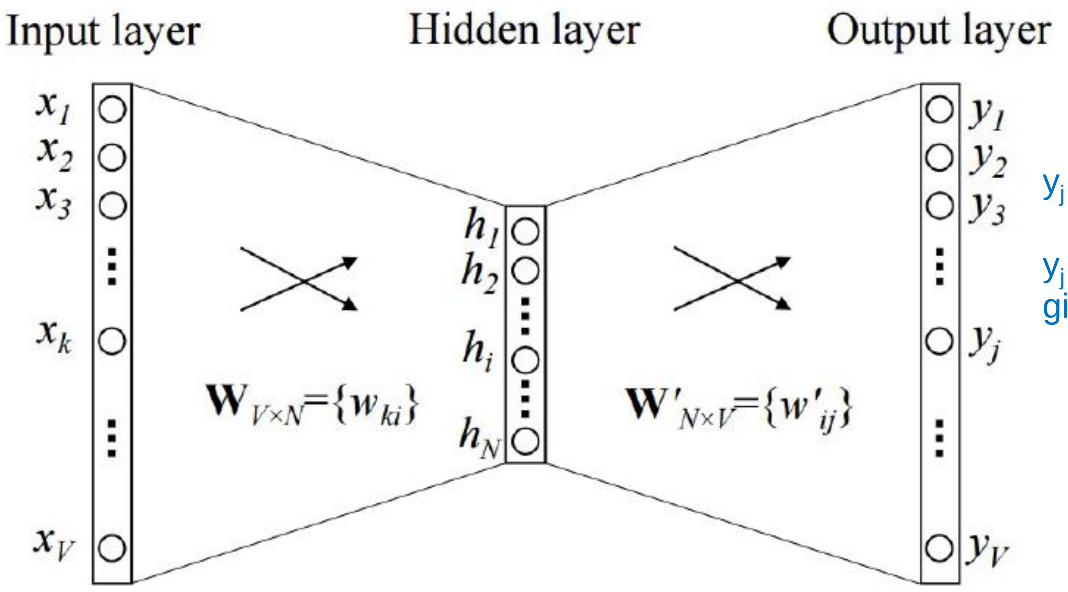
$$v'^{new}_{w_j} = v'^{old}_{w_j} - \alpha \frac{1}{C} \frac{\partial E}{\partial h} \forall j \in 1, \dots, C$$ where $\frac{\partial E}{\partial h} = \left(\frac{\partial E}{\partial h_1}, \dots, \frac{\partial E}{\partial h_N} \right)$ and a is the learning rate of the gradient descent. The derivative $\frac{\partial E}{\partial u_j} = y_j - \delta_{j, j^M}$ represents the error of the network with respect to the true target word so that the error is back propagated on the system, which can learn iteratively. Note that the vectors v_{w_j} $\forall j 1, .., |V|$ are the usual word representations used to perform the semantic operations.

Further details can be found in the Rong (2015) paper.

5.4 A very simple continuous bag of words model

- Assume that **c = 1**. i.e. The length of the window is 1.
- Predict one target word, given one context word.





y_j = Probability (word$_j$ | word context) for j = 1...V

y_j is the probability of observing word$_j$ given the word context

One-hot representation vector

$$X = \begin{matrix} 0 \\ 0 \\ 1 \\ . \\ . \\ . \\ 0 \end{matrix}$$

$$H = W^T X$$

$$U = W'^T H$$

$$y_j = \frac{e^{u_j}}{\sum_{i=1}^{|V|} e^{u_i}}$$

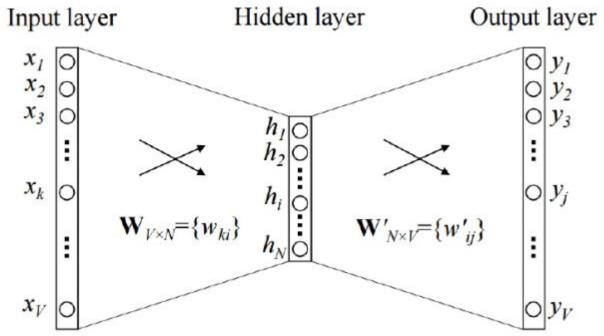
Softmax function

5.5 Example

- We will take a very small set of the corpus to understand the concept. See the sentences from our small corpus given as follows:
  - the dog saw a cat
  - the dog chased a cat
  - the **cat climbed** a tree
- The preceding three sentences have eight (8) unique words. We need to order them in alphabetical order, and if we want to access them, then we will refer to the index of each word. Refer to the following table :

- Here our value for **V** is equal to **8 : V=8**
- Let's assume we will have three (3) neurons

in the hidden layer: **N=3**



Words	Index
a	1
cat	2
chased	3
climbed	4
dog	5
saw	6
the	7
tree	8

5.5 Example (continued)

- Before training begins, these matrices, **W** and **W’** , are initialized by using small random values, as is very common in neural network training. Just for illustration purposes, let us assume that **W** and **W’** are to be initialized to the following values:

$$
W = \begin{matrix}
-0.094491 & -0.443977 & 0.313917 \\
-0.490796 & -0.229903 & 0.065460 \\
0.072921 & 0.172246 & -0.357751 \\
0.104514 & -0.463000 & 0.079367 \\
-0.226080 & -0.154659 & -0.038422 \\
0.406115 & -0.192794 & -0.441992 \\
0.181755 & 0.088268 & 0.277574 \\
-0.055334 & 0.491792 & 0.263102
\end{matrix}
$$

$$
W' = \begin{matrix}
0.023074 & 0.479901 & 0.432148 & 0.375480 & -0.364732 & -0.119840 & 0.266070 & -0.351000 \\
-0.368008 & 0.424778 & -0.257104 & -0.148817 & 0.033922 & 0.353874 & -0.144942 & 0.130904 \\
0.422434 & 0.364503 & 0.467865 & -0.020302 & -0.423890 & -0.438777 & 0.268529 & -0.446787
\end{matrix}
$$

5.5 Example (continued)

- We are targeting it so that our neural network can learn the relationship between the words **cat** and **climbed**. So, in other words, we can explain that the neural network should give high probability for the word **climbed** when the word **cat** is fed into the neural network as an input. So:
  - context word = “**cat**”
  - target word = “**climbed**”

The input vector X *(for the first layer)* stands for the word **cat** and it will be :

$$X = \begin{matrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

On-hot representation vector for the word **cat**

5.5 Example (continued)

- The hidden layer output ***H*** is calculated by using the following formula:

$$H = W^T X = \begin{matrix} -0.490796 \\ -0.229903 \\ 0.065460 \end{matrix}$$

From the preceding calculation, we can figure out that, here, the output of the hidden neurons mimics the weights of the second row of the **W** matrix because of one-hot encoding representation.

5.5 Example (continued)

- Computing the output layer **Y**:

From H to output layer, we have to multiply W'^T to H and then apply a non-linear function on that to get **Y**. In word2vec, we are converting activation values of the output layer to probabilities by using the Softmax function.

$$U = W'^T H = \begin{matrix} 0.100934 \\ -0.309331 \\ -0.122361 \\ -0.151399 \\ 0.143463 \\ -0.051262 \\ -0.079686 \\ 0.112928 \end{matrix}$$

5.5 Example (continued)

- Here, our final goal is to obtain probabilities for words in the output layer. From the output layer, we are generating probability that reflects the next word relationship with the context word at input. Thus, the mathematical representation is given as follows:

y_j = Probability (word_j | word context) for j = 1...V

Basically y_j is the probability of observing word_j given the word context:

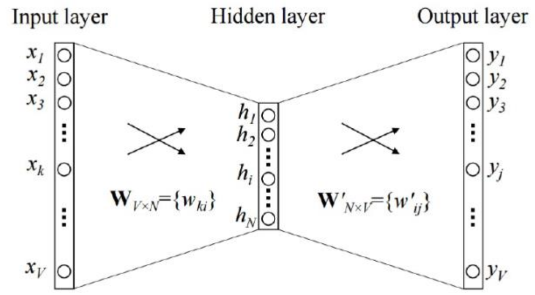
$$y_j = \frac{e^{u_j}}{\sum_{i=1}^{|V|} e^{u_i}}$$

5.5 Example (continued)

- By using this equation, we can calculate the probabilities for eight words in the corpus and the probability values are given as follows:

On-hot vector for the word **cat**

X = $\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$


Input layer
Hidden layer
Output layer
x_1 x_2 x_3 x_k x_V
h_1 h_2 h_i h_N
y_1 y_2 y_3 y_j y_V
$W_{V\times N}=\{w_{ki}\}$
$W'_{N\times V}=\{w'_{ij}\}$

y = $\begin{bmatrix}0.143073\\0.094925\\0.114441\\\mathbf{0.111166}\\0.149289\\0.122874\\0.119431\\0.144800\end{bmatrix}$

The **true** target vector (word **climbed**) is: $\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}$

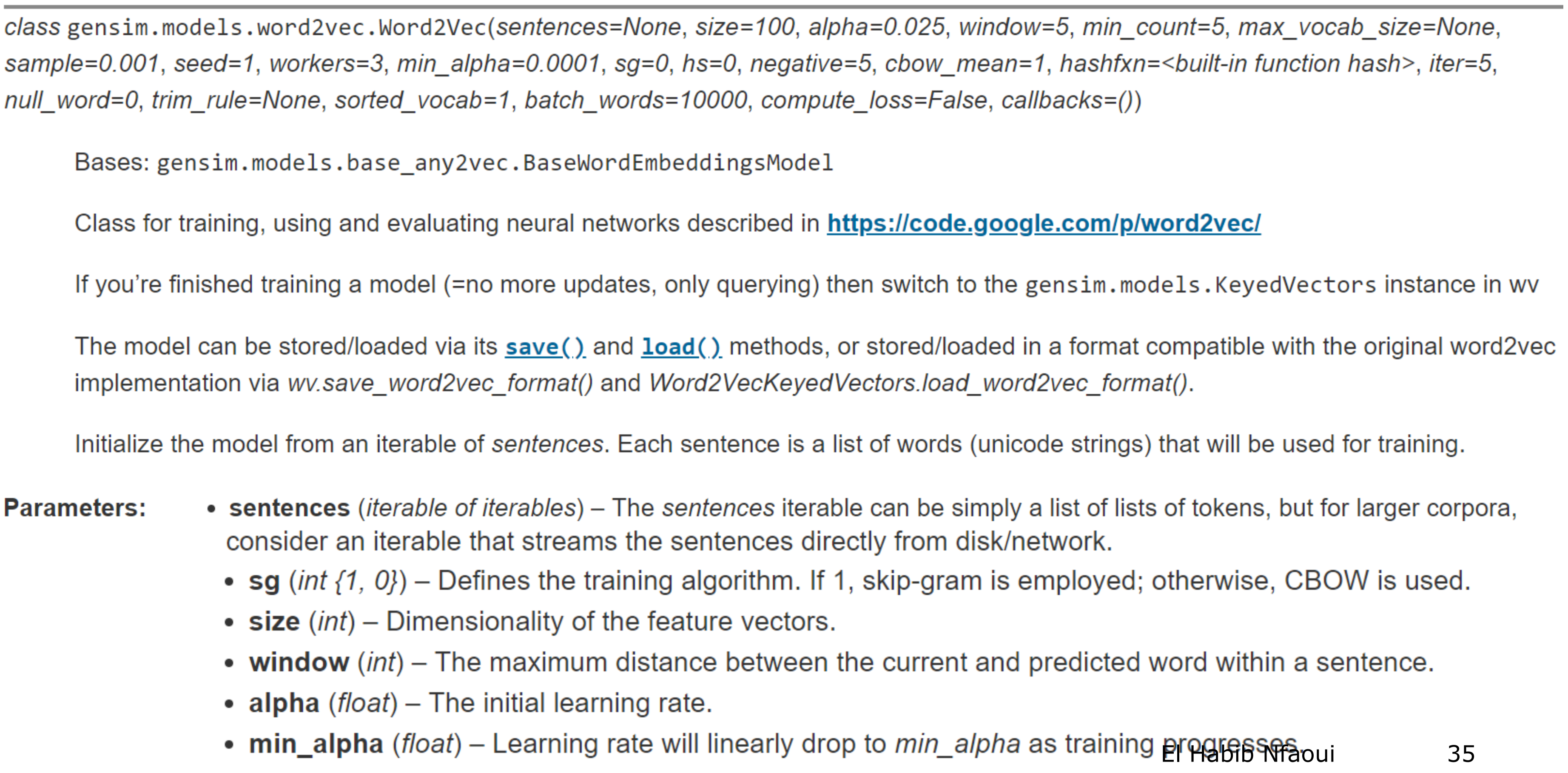
As you can see, the probability **0.111166** is for the chosen target word **climbed**. As we know, the target vector is *[0 0 0 1 0 0 0 0]^T*, so we can compute the error by prediction. To generate a prediction error or error vector, we need to subtract the probability vector from the target vector, and once we know the error vector or error values, we can adjust the weight according to that. Here, we need to adjust the weight values of the matrices *W* and *W '*. The technique of propagating errors in the network and readjusting the weight values of *W* and *W '* is called **backpropagation**.

Thus, the training can continue by taking different context-target word pairs from the corpus. This is the way word2vec learns relationships between words in order to develop a vector representation of words in the corpus.

5.6 Implementation: gensim Python Library

models.word2vec – word2vec

https://radimrehurek.com/gensim/models/word2vec.html



class `gensim.models.word2vec.Word2Vec`(*sentences=None*, *size=100*, *alpha=0.025*, *window=5*, *min_count=5*, *max_vocab_size=None*, *sample=0.001*, *seed=1*, *workers=3*, *min_alpha=0.0001*, *sg=0*, *hs=0*, *negative=5*, *cbow_mean=1*, *hashfxn=<built-in function hash>*, *iter=5*, *null_word=0*, *trim_rule=None*, *sorted_vocab=1*, *batch_words=10000*, *compute_loss=False*, *callbacks=()*)

Bases: `gensim.models.base_any2vec.BaseWordEmbeddingsModel`

Class for training, using and evaluating neural networks described in **https://code.google.com/p/word2vec/**

If you're finished training a model (=no more updates, only querying) then switch to the `gensim.models.KeyedVectors` instance in wv

The model can be stored/loaded via its **`save()`** and **`load()`** methods, or stored/loaded in a format compatible with the original word2vec implementation via *wv.save_word2vec_format()* and *Word2VecKeyedVectors.load_word2vec_format()*.

Initialize the model from an iterable of *sentences*. Each sentence is a list of words (unicode strings) that will be used for training.

Parameters:

- **sentences** (*iterable of iterables*) – The *sentences* iterable can be simply a list of lists of tokens, but for larger corpora, consider an iterable that streams the sentences directly from disk/network.
- **sg** (*int {1, 0}*) – Defines the training algorithm. If 1, skip-gram is employed; otherwise, CBOW is used.
- **size** (*int*) – Dimensionality of the feature vectors.
- **window** (*int*) – The maximum distance between the current and predicted word within a sentence.
- **alpha** (*float*) – The initial learning rate.
- **min_alpha** (*float*) – Learning rate will linearly drop to *min_alpha* as training progresses.

5.6 Implementation: gensim Python Library

- **iter** (*int*) – Number of epochs over the corpus.
- **min_count** (*int*) – Ignores all words with total frequency lower than this.
- **batch_words** (*int*) – Target size (in words) for batches of examples passed to worker threads (and thus cython routines). (Larger batches will be passed if individual texts are longer than 10000 words, but the standard cython code truncates to that maximum.)

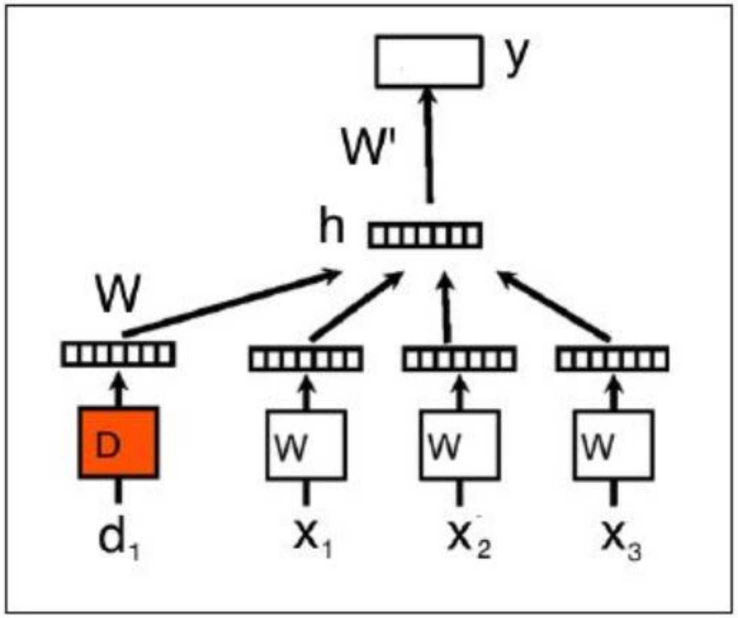
5.7 Doc2Vec extension

As explained in Le and Mikolov (2014), Doc2Vec is a natural extension of the word2vec method in which a document is considered as an additional word vector. So in the case of the CBOW architecture, the hidden layer vector h is just the average of the context vectors and the document vector d_i:

$$h = \frac{1}{C} W \cdot (x_1 + .. + x_C + d_i) = \frac{1}{C}\left(v_{w_1} + .. + v_{w_c} + v_{d_i}\right) = v_C$$

This architecture is shown in the following figure, and it is called the **distributed memory model** (**DM**) because the document d_i vector just remembers the information of the document not represented by the context words. The vector v_{d_i} is shared with all the context words sampled from the document d_i but the matrix W (and W') is the same for all the documents:

5.7 Doc2Vec extension



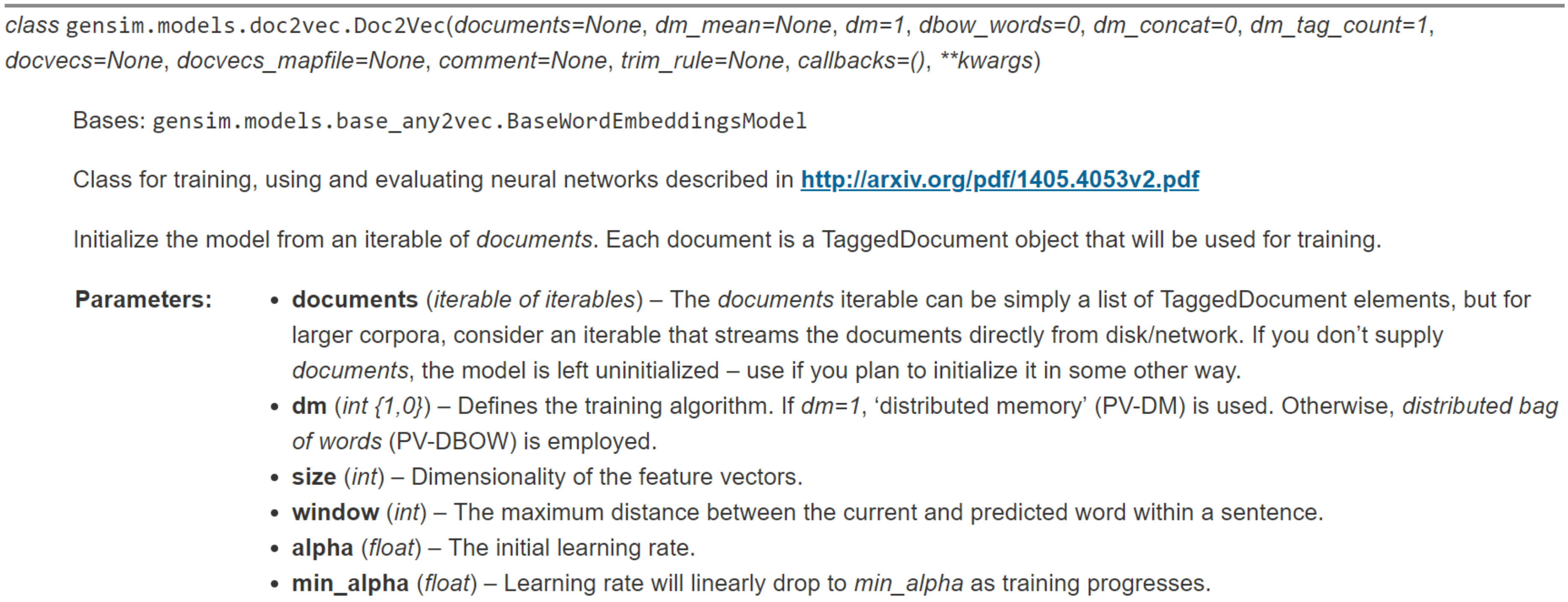
A distributed memory model example with a context of three words (window=3); figure taken from Distributed Representations of Sentences and Documents by Le and Mikolov (2014)

The other proposed architecture is called **distributed bag of words (DBOW)**, which considers only a document vector in the input layer and a set of context words sampled from the document in the output layer. It has been shown that the DM architecture performs better than DBOW, and it is therefore the default model in the Python **gensim library** implementation. The reader is advised to read the paper of Le and Mikolov (2014) for further details.

5.8 Implementation: gensim Python Library

models.doc2vec

https://radimrehurek.com/gensim/models/doc2vec.html


class `gensim.models.doc2vec.Doc2Vec`(*documents=None*, *dm_mean=None*, *dm=1*, *dbow_words=0*, *dm_concat=0*, *dm_tag_count=1*, *docvecs=None*, *docvecs_mapfile=None*, *comment=None*, *trim_rule=None*, *callbacks=()*, ***kwargs*)

Bases: `gensim.models.base_any2vec.BaseWordEmbeddingsModel`

Class for training, using and evaluating neural networks described in http://arxiv.org/pdf/1405.4053v2.pdf

Initialize the model from an iterable of *documents*. Each document is a TaggedDocument object that will be used for training.

Parameters:

- **documents** (*iterable of iterables*) – The *documents* iterable can be simply a list of TaggedDocument elements, but for larger corpora, consider an iterable that streams the documents directly from disk/network. If you don't supply *documents*, the model is left uninitialized – use if you plan to initialize it in some other way.
- **dm** (*int {1,0}*) – Defines the training algorithm. If *dm=1*, 'distributed memory' (PV-DM) is used. Otherwise, *distributed bag of words* (PV-DBOW) is employed.
- **size** (*int*) – Dimensionality of the feature vectors.
- **window** (*int*) – The maximum distance between the current and predicted word within a sentence.
- **alpha** (*float*) – The initial learning rate.
- **min_alpha** (*float*) – Learning rate will linearly drop to *min_alpha* as training progresses.

6. Latent Semantic Analysis (LSA)

The name of this algorithm comes from the idea that there is a latent space in which each word (and each document) can be efficiently described, assuming that words with similar meanings also occur in similar text positions. Projection on this subspace is performed using the (truncated) SVD (Singular value decomposition) method.

We contextualize the method for LSA as follows: the web pages are collected together in matrix X (V ´N), in which each column is a document:

$$X = U_t \Sigma_t V_t^T$$

Here, U_t $(V \, ´d)$ is the matrix of the words projected in the new latent space with d dimensions, $\Sigma_t V_t^T$ $(d \, ´N)$ is the transpose matrix of the documents transformed into the subspace, and Σ_t $(d´d)$ is the diagonal matrix with singular values. The query vector itself is projected into the latent space by:

$$q_t = q U_t \Sigma_t^{-1}$$

6. Latent Semantic Analysis (LSA)

Now, each document represented by each row of V_t can be compared with q_t using the cosine similarity. Note that the true mathematical representation of the documents on the latent space is given by $\Sigma_t V_t$ (not V_t) because the singular values are the scaling factors of the space axis components and they must be taken into account. Therefore, this matrix should be compared with $\Sigma_t q_t$. Nevertheless, it usually computes the similarity between V_t and q_t, and in practice, it is still unknown which method returns the best results.

6.1 Discussion

- LSI has been shown to perform better than traditional keywords based methods. **The main drawback is the time complexity of the SVD, which is $O(m^2n)$. It is thus difficult to use for a large document collection such as the Web**. Another drawback is that the concept space is not interpretable as its description consists of all numbers with little semantic meaning.

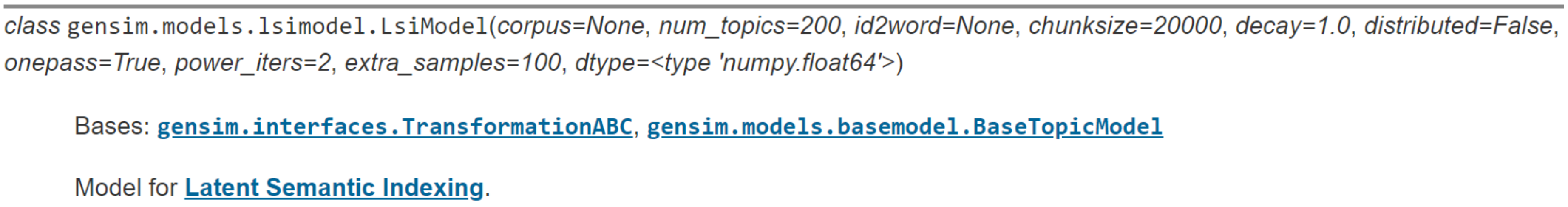
  Determining the optimal number of dimensions k of the concept space is also a major difficulty. There is no general consensus for an optimal number of dimensions. The original paper [18] of LSI suggests 50–350 dimensions. In practice, the value of k needs to be determined based on the specific document collection via trial and error, which is a very time consuming process due to the high time complexity of the SVD.

6.2 Implementation: gensim Python Library

models.lsimodel – Latent Semantic Indexing

Module for Latent Semantic Analysis (aka Latent Semantic Indexing).

https://radimrehurek.com/gensim/models/lsimodel.html


class gensim.models.lsimodel.LsiModel(*corpus=None*, *num_topics=200*, *id2word=None*, *chunksize=20000*, *decay=1.0*, *distributed=False*, *onepass=True*, *power_iters=2*, *extra_samples=100*, *dtype=<type 'numpy.float64'>*)

Bases: gensim.interfaces.TransformationABC, gensim.models.basemodel.BaseTopicModel

Model for Latent Semantic Indexing.

7. Information retrieval system evaluation: Evaluation Measures

- We have seen in the preceding chapters many alternatives in designing an IR system. How do we know which of these techniques are effective in which applications?

  To measure information retrieval effectiveness in the standard way, we need a test collection consisting of three elements:

  1. A document collection
  2. A test suite of information needs, expressible as queries.
  3. A set of relevance judgments, standardly a binary assessment of either relevant or irrelevant for each query-document pair.

7. Information retrieval system evaluation: Evaluation Measures

The standard approach to information retrieval system evaluation revolves around the notion of *relevant* and *nonrelevant* documents. With respect to a user information need, a document in the test collection is given a **binary classification as either relevant or nonrelevant**. This decision is referred to as the *gold standard* or *ground truth* judgment of relevance. The test document collection and suite of information needs have to be of a **reasonable size**: you need to average performance over fairly **large test sets**, as results are highly variable over different documents and information needs. As a rule of thumb, **50 information needs has usually been found to be a sufficient minimum**. Because **from this minimum**, we can make (use) a **Statistical significance test** (such as Student test, Wilcoxon test, etc.). A result of an experiment is said to have statistical significance, or **be statistically significant**, if it is likely not caused by chance for a given statistical significance level. Your statistical significance level reflects your risk tolerance and confidence level. For example, if you run an A/B testing experiment with a significance level of 95%, this means that if you determine a winner, you can be 95% confident that the observed results are real and not an error caused by randomness. It also means that there is a 5% chance that you could be wrong.

7.1 Standard test collections

- *TREC: Text Retrieval Conference (TREC)*. The U.S. National Institute of Standards and Technology (NIST) has run a large IR test bed evaluation series since 1992.
- CLEF: Cross Language Evaluation Forum (*CLEF*)
- NTCIR: NII Test Collections for IR Systems (*NTCIR*)
- …..

7.2 Evaluation of unranked retrieval sets

- How is IR system effectiveness measured?

The two most frequent and basic measures for information retrieval effectiveness are **precision** and **recall**. These are first defined for the simple case where an IR system returns a set of documents for a query. We will see later how to extend these notions to ranked retrieval situations.

Recall

A measure of the ability of a system to present all relevant items.

$$\text{recall} = \frac{\text{number of relevant items retrieved}}{\text{number of relevant items in collection}}$$

recall Є [0,1]

Precision.

A measure of the ability of a system to present only relevant items.

$$\text{precision} = \frac{\text{number of relevant items retrieved}}{\text{total number of items retrieved}}$$

precision Є [0,1]

Precision and recall are set based measures. That is they evaluate the quality of an unranked (unordered) set of retrieved documents.

7.2 Evaluation of unranked retrieval sets

- These notions can be made clear by examining the following contingency table:

	Relevant	Nonrelevant
Retrieved	true positives (tp)	false positives (fp)
Not retrieved	false negatives (fn)	true negatives (tn)

Then:

$$\begin{aligned} P &= tp/(tp+fp) \\ R &= tp/(tp+fn) \end{aligned}$$

7.2 Evaluation of unranked retrieval sets

The advantage of having the two numbers for precision and recall is that one is more important than the other in many circumstances. Typical web surfers would like every result on the first page to be relevant (**high precision**) but have not the slightest interest in knowing let alone looking at every document that is relevant. In contrast, various professional searchers such as paralegals and intelligence analysts are very concerned with trying to get as high recall as possible, and will tolerate fairly low precision results in order to get it. Individuals searching their hard disks are also often interested in **high recall** searches. Nevertheless, the two quantities clearly trade off against one another: you can always get a recall of 1 (but very low precision) by retrieving all documents for all queries! Recall is a non-decreasing function of the number of documents retrieved. On the other hand, in a good system, precision usually decreases as the number of documents retrieved is increased.

Although in theory precision and recall do not depend on each other, in practice a high recall is almost always achieved at the expense of precision, and a high precision is achieved at the expense of recall. Thus, precision and recall has a trade-off. Depending on the application, one may want a high precision or a high recall.

If we need a single measure to compare different systems, the **F-score** is often used:

7.2 Evaluation of unranked retrieval sets

$$F = \frac{2pr}{p+r}$$

The F-score (also called the **F_1-score**) is the harmonic mean of precision and recall (**it equally weights precision and recall**).

$$F = \frac{2}{\frac{1}{p}+\frac{1}{r}}$$

The harmonic mean of two numbers tends to be closer to the smaller of the two. Thus, for the F-score to be high, both *p* and *r* must be high.

- Recall, precision, and the F-score are inherently measures between 0 and 1, but they are also very commonly written **as percentages**, on a scale between 0 and 100.

Example

- Consider the following search results for two queries Q1 and Q2 (the documents are unranked, the relevant documents are shown in bold).

  Q1: D1, **D2**, D3, **D4**, **D5**.

  Q2: **D1**, D2, **D3**.

For Q1 and Q2 the total number of relevant documents in the collection (corpus) is, respectively, 10 and 6.

For each query, calculate the precision and the recall values.

Solution:

For query Q1:

number of relevant documents retrieved: 3

number of relevant documents in collection: 10

Total number of documents retrieved: 5

$\Rightarrow$ **recall= 3/10**
precision = 3/5

For query Q2: …………

7.3 Evaluation of ranked retrieval results

- Precision, recall, and the F-measure are set-based measures. They are computed using unordered sets of documents. We need to extend these measures (or to define new measures) if we are to evaluate the ranked retrieval results that are now standard with search engines.

- In IR and Web search, a ranking of the documents is produced for the user. This section studies how to evaluate such rankings.

Let the collection of documents in the database be *D*, and the total number of documents in *D* be *N*. Given a user query **q**, the retrieval algorithm first computes relevance scores for all documents in *D* and then produce a ranking R_q of the documents based on the relevance scores, i.e.,

$$R_q : \ < \mathbf{d}_1^q, \mathbf{d}_2^q, \ldots, \mathbf{d}_N^q >$$

a. Precision@K : Precision at rank position k

where $\mathbf{d}_1^q \in D$ is the most relevant document to query **q** and $\mathbf{d}_N^q \in D$ is the most irrelevant document to query **q**.

Let D_q ($\subseteq D$) be the set of actual relevant documents of query **q** in D. We can compute the precision and recall values at each $\mathbf{d}_i^q$ in the ranking.

Recall at rank position i or document $\mathbf{d}_i^q$ (denoted by $r(i)$) is the fraction of relevant documents from $\mathbf{d}_1^q$ to $\mathbf{d}_i^q$ in R_q. Let the number of relevant documents from $\mathbf{d}_1^q$ to $\mathbf{d}_i^q$ in R_q be s_i ($\leq |D_q|$) ($|D_q|$ is the size of D_q). Then,

$$r(i) = \frac{s_i}{|D_q|}$$

(can be also denoted by R@i or "Recall at i")

Precision at rank position i or document $\mathbf{d}_i^q$ (denoted by $p(i)$) is the fraction of documents from $\mathbf{d}_1^q$ to $\mathbf{d}_i^q$ in R_q that are relevant:

$$p(i) = \frac{s_i}{i}$$

(can be also denoted by P@i or "Precision at i")

a. Precision@K (Continued)

Another often used evaluation measure is the **F-score**. Here we can compute the **F-score** at each **rank position *i***. Recall that F-score is the harmonic mean of precision and recall:

$$F(i) = \frac{2}{\frac{1}{r(i)} + \frac{1}{p(i)}} = \frac{2p(i)r(i)}{p(i) + r(i)}$$

Example 1:

- We have a document collection *D* with 20 documents. Given a query **q**, we know that eight documents are relevant to **q**. A retrieval algorithm produces the ranking (of all documents in *D*) shown in the table below.

In column 1, 1 represents the highest rank and 20 represents the lowest rank. "+" and "-" in column 2 indicate a relevant document and an irrelevant document respectively. The precision (P@i) and recall (R@i) values at each position i are given in columns 3 and 4.

Rank *i*	+/–	P@i	R@i
1	+	1/1 = 100%	1/8 = 13%
2	+	2/2 = 100%	2/8 = 25%
3	+	3/3 = 100%	3/8 = 38%
4	–	3/4 = 75%	3/8 = 38%
5	+	4/5 = 80%	4/8 = 50%
6	–	4/6 = 67%	4/8 = 50%
7	+	5/7 = 71%	5/8 = 63%
8	–	5/8 = 63%	5/8 = 63%
9	+	6/9 = 67%	6/8 = 75%
10	+	7/10 = 70%	7/8 = 88%
11	–	7/11 = 63%	7/8 = 88%
12	–	7/12 = 58%	7/8 = 88%
13	+	8/13 = 62%	8/8 = 100%
14	–	8/14 = 57%	8/8 = 100%
15	–	8/15 = 53%	8/8 = 100%
16	–	8/16 = 50%	8/8 = 100%
17	–	8/17 = 53%	8/8 = 100%
18	–	8/18 = 44%	8/8 = 100%
19	–	8/19 = 42%	8/8 = 100%
20	–	8/20 = 40%	8/8 = 100%

b. Average Precision

Average Precision: Sometimes we want a single precision to compare different retrieval algorithms on a query **q**. An average precision (p_{avg}) can be computed based on the precision at each relevant document in the ranking,

Only the relevant document

$$p_{avg} = \frac{\sum_{d_i^q \in D_q} p(i)}{|D_q|}$$

Example :

For the ranking in the previous table of Example 1, the average precision is 81%:

$$p_{avg} = \frac{100\% + 100\% + 100\% + 80\% + 71\% + 67\% + 70\% + 62\%}{8} = 81\%.$$

c. Mean Average Precision

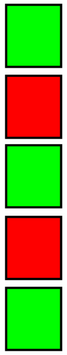
- For a single information need (query), Average Precision is the average of the precision value obtained for the set of top *k* documents existing after each relevant document is retrieved, and this value is then averaged over **information needs**. That is, if the set of relevant documents for an information need $q_j \in Q$ is $\{d_1, \ldots d_{m_j}\}$ and R_{jk} is the set of ranked retrieval results from the top result until you get to document d_k, then

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk})$$

Using MAP, **fixed recall levels are not chosen**, and there is **no interpolation (see section Precision-Recall curve)**.

c. Mean Average Precision

Calculating MAP: Summary

- Consider rank position of each ***relevant*** doc
  - $K_1, K_2, \ldots K_R$
- Compute Precision@K for each $K_1, K_2, \ldots K_R$
- Average precision = average of P@K
- Ex:  has AvgPrec of

$$\frac{1}{3} \cdot \left(\frac{1}{1} + \frac{2}{3} + \frac{3}{5} \right) \approx 0.76$$

- MAP is **Average Precision** across **multiple queries/rankings.**

Example : Average Precision

■ ■ ■ ■ ■ ■ = the relevant documents

Ranking #1	■	□	■	■	■	■	□	□	□	■
Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

Ranking #2	□	■	□	□	■	■	■	□	■	■
Recall	0.0	0.17	0.17	0.17	0.33	0.5	0.67	0.67	0.83	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57	0.5	0.56	0.6

$$\text{Ranking \#1: } (1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$$

$$\text{Ranking \#2: } (0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6)/6 = 0.52$$

Example: MAP

■ ■ ■ ■ ■ = relevant documents for query 1

Ranking #1	■	□	■	□	□	■	□	□	■	■
Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5

■ ■ ■ = relevant documents for query 2

Ranking #2	□	■	□	□	■	□	■	□	□	□
Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3

$$\textit{average precision query 1} = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$$
$$\textit{average precision query 2} = (0.5 + 0.4 + 0.43)/3 = 0.44$$

$$\textit{mean average precision} = (0.62 + 0.44)/2 = 0.53$$

c. Mean Average Precision

- Among evaluation measures, MAP has been shown to have especially good discrimination and stability.
- MAP assumes user is interested in finding many relevant documents for each query.
- MAP requires many relevance judgments in text collection.

d. R-Precision

- **R-Precision** is the precision after **R** documents have been retrieved, where **R** is the **number of relevant documents for the topic**.

- The **average R-Precision** for a run is computed by taking the mean of the R-Precisions of the individual topics in the run. For example, assume a run consists of two topics, one with 50 relevant documents and another with 10 relevant documents. If the retrieval system returns 17 relevant documents in the top 50 documents for the first topic, and 7 relevant documents in the top 10 for the second topic, then the run's R-Precision would be

$$\frac{\frac{17}{50} + \frac{7}{10}}{2} \text{ or } 0.52.$$

e. Precision–Recall Curve

Based on the precision and recall values at each rank position, we can draw a precision–recall curve where the x-axis is the recall and the y-axis is the precision. Instead of using the precision and recall at each rank position, the curve is commonly plotted using 11 standard recall levels, 0.0, 0.1, 0.2, . . . , 1.0.

Since we may not obtain exactly these recall levels in the ranking, interpolation is needed to obtain the precisions at these recall levels, which is done as follows:

The ***interpolated precision*** p_{interp} at a certain recall level r is defined as the highest precision found for any recall level $r' \geq r$:

$$p_{interp}(r) = \max_{r' \geq r} p(r')$$

- **Postscript: TREC Interpolation Rule (TREC-6 Appendix A)**

*For each **standard** recall value 'i' from 0.0 to 1.0 with 0.1 increments, take the maximum precision obtained at any **actual** recall value greater or equal to 'i'.*

e.1 Exercise

- Consider the following search result for a query Q1 (the documents are ranked in the given order, the relevant documents are shown in bold).

  Q1: **D1**, D2, **D3**, D4, **D5**, **D6**, D7, D8, D9, D10.

  For Q1 the total number of relevant documents is 4.

  Using the TREC interpolation rule, in a table give the precision value for the 11 standard recall levels 0.0, 0.1, 0.2, … 1.0. Please also draw the corresponding recall-precision graph as shown in the first figure of TREC-6 Appendix A.

Solution

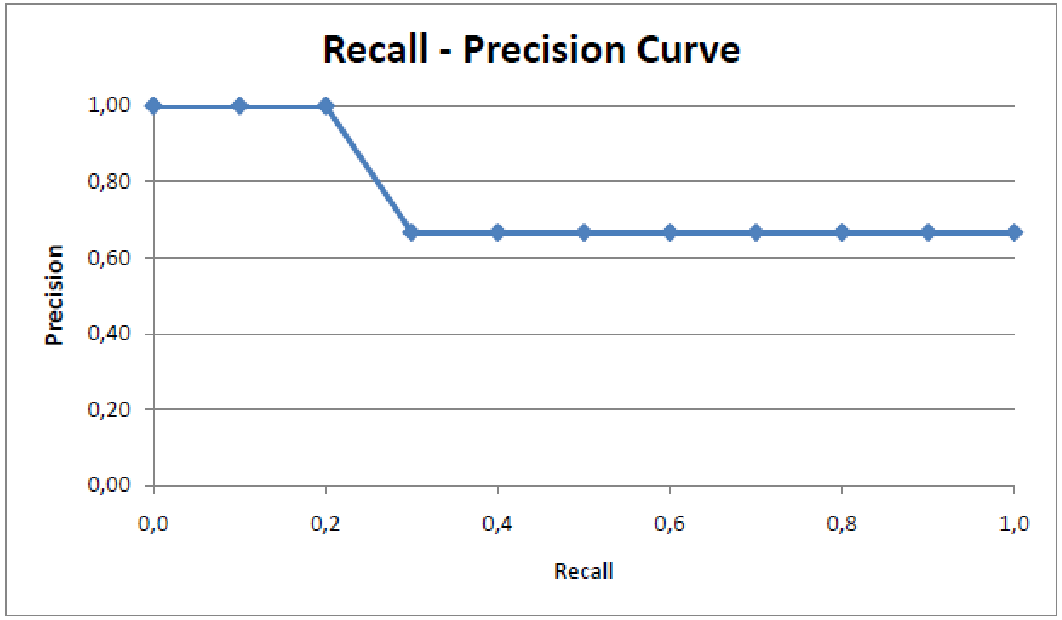
Actual Recall & Precision Table:

Rank	1	2	3	4	5	6	7	8	9	10
Relevant	Yes	No	Yes	No	Yes	Yes	No	No	No	No
Precision	1/1	1/2	2/3	2/4	3/5	4/6	4/7	4/8	4/9	4/10
Recall	0.25	0.25	0.50	0.50	0.75	1.00	1.00	1.00	1.00	1.00

Interpolated Recall & Precision Table:

Precision	1	1	1	2/3	2/3	2/3	4/6 = 2/3	4/6	4/6	4/6	4/6
Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Recall Precision Graph:



e.2 Comparing Different Algorithms

- Frequently, we need to compare the retrieval results of different algorithms. We can draw their precision-recall curves together in the same figure for comparison. Figure below shows the curves of two algorithms on the same query and the same document collection. We observe that the precisions of one algorithm are better than those of the other at low recall levels, but are worse at high recall levels.

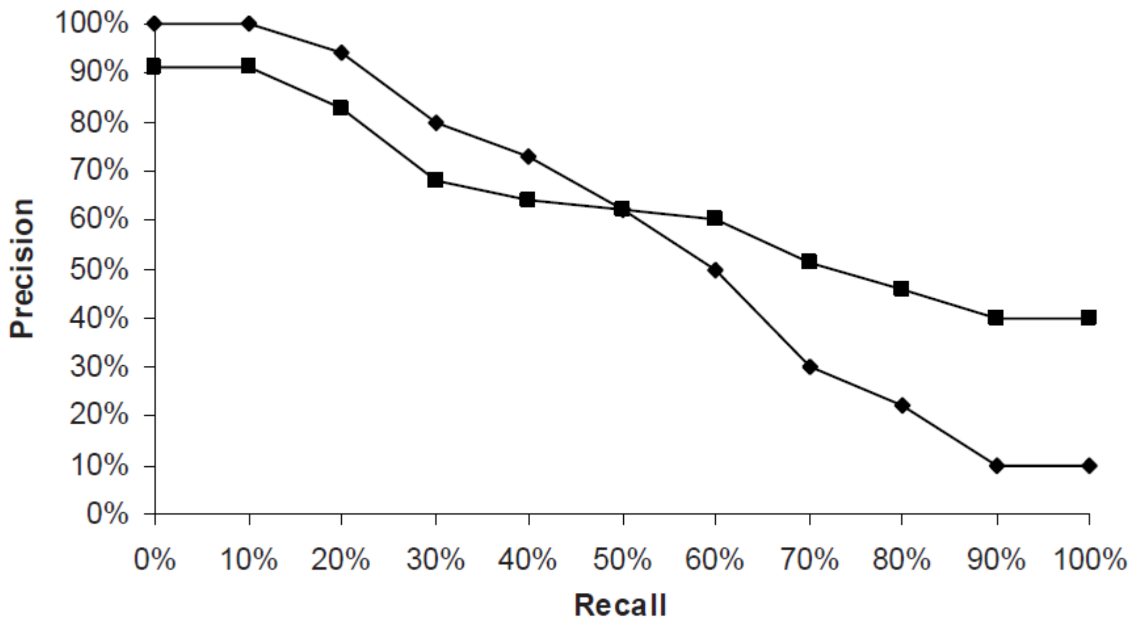


Figure . Comparison of two retrieval algorithms based on their precision-recall curves

e.3 Evaluation Using Multiple Queries

In most retrieval evaluations, we are interested in the performance of an algorithm on a large number of queries.

The overall precision (denoted by $\overline{p}(r_i)$) at each recall level r_i is computed as the average of individual precisions at that recall level, i.e.,

$$\overline{p}(r_i) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} p_j(r_i)$$

where Q is the set of all queries and $p_j(r_i)$ is the precision of query j at the recall level r_i. Using the average precision at each recall level, we can also draw a precision-recall curve.

f. Other measures

- We should note that precision is not the only measure for evaluating search ranking. Reputation or quality of the top ranked pages is also very important.

  Finally, the precision and recall **breakeven point** is also a commonly used measure.

g. Discussion

- One problem with precision and recall measures is that, in many applications, it can be very hard to determine the set of **relevant documents D_q** for each query **q**. For example, **on the Web**, D_q is almost impossible to determine because there are simply too many pages to manually inspect. Without D_q, the recall value cannot be computed. In fact, recall does not make much sense for Web search because the user seldom looks at pages ranked below 30. However, precision is critical, and it can be estimated for top ranked documents. Manual inspection of only the top 30 pages is reasonable. The following precision computation is commonly used:

  We compute the precision values at some selected rank positions. For a Web search engine, we usually compute precisions for the top **5, 10, 15, 20, 25 and 30 returned pages** (as the user seldom looks at more than 30 pages). We assume that the number of relevant pages is more than 30. Following Example 1, we have

  p(5) = 80%, p(10) = 70%, p(15) = 53%, and p(20) = 40%.

References

Bing Liu. Web Data Mining. Pub. Date: 2011, Second Edition, pages: 622. ISBN: 978-3-642-19459-7. Publisher: Springer-Verlag Berlin Heidelberg

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.

Christopher D. Manning. Lecture slides: Introduction to Information Retrieval

Diana Inkpen, Information Retrieval and the Internet. Lecture slides. http://www.site.uottawa.ca/~diana/csi4107/

Fazli Can. CS533 course: Information Retrieval Systems, Lecture slides. Computer Engineering Department, Bilkent University

Isoni Andrea. Machine Learning for the Web. Pub. Date: 2016, pages: 299, ISBN: 978-1-78588-660-7. Publisher: Packt Publishing

Jaideep Srivastava. Web Mining : Accomplishments & Future Directions, University of Minnesota, USA

Jalaj Thanaki. Python Natural Language Processing. Pub. Date: 2017, ISBN 978-1-78712-142-3. Packt Publishing Ltd.

Salton, G. and C. Buckley. Term-weighting approaches in automatic text retrieval. Information Processing & Management, 1988, 24(5): p. 513-523.

TREC Appendix: https://trec.nist.gov/pubs/trec15/appendices/CE.MEASURES06.pdf

https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285