# Week 2 Review and Questions

## 1. Processing Text

### From Words to Terms and Text Statistics

The first step for text analysis is to manipulate the text in some way to conduct information retrieval or text analysis. Text processing is a way to find useful index 'terms' or text features from words.

There are several issues that you need to consider when you decide the terms. The first one is case sensitivity since people normally do not want a search function that can only do exactly matching. For example, most search engines do not distinguish between uppercase and lowercase letters. The second one is that not all words are of equal value in a search or text analysis. Usually, *tokenization* is used to split words. Some words, or call them stopping, may be ignored entirely. We may also use stemming to allow similar words (like "run" and "running") to have the same semantic meaning. For Web documents, such as web pages, may include formatting information (like bold or large text), explicit structure (like XML tags: <title>, <chapter>, and <text>), and/or hyperlinks.

Text processing may be profound to the results of text analysis. These text processing techniques may be simple, but it can take a lot of time for data engineers to produce high quality terms or text features. Understanding the statistical nature of text is fundamental to understanding the meaning of high-quality terms or text features.

Statistical models of word occurrences are very important in IR (Information Retrieval). One of the interesting observations is that the distribution of word frequencies is very skewed. That means there are a few words that have very high frequencies and many words that have low frequencies.

**Question 1**. Please open a text or an XML file (e.g., *6146.xml*) and represent it as a list of paragraphs or sentences, *text*. You may remove any non-relevant information (e.g., ' ', ' ', '\n'). After that, you need to find all terms and their frequencies (the number of occurrences in the file) if their length > 2, represent them using a dictionary, *doc*; and print the number of total terms (e.g., 137). Then print the top-10 terms in *doc* in a descending order, e.g.,

[('the', 8), ('technical', 2), ('bounce', 2), ('said', 2), ('and', 2), ('not', 2), ('due', 2), ('rose', 2), ("argent ina's", 2), ('argentine', 1)]

At last, please plot the distribution of the top-10 terms (see, Fig 1 for an example).

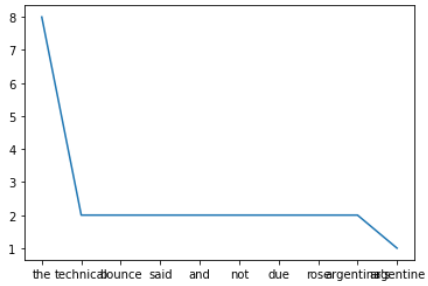


**Fig 1**. Example of distribution of top-k frequent words.

For a very large data collection or corpus, Zipf's law is possible true to predict the relationship between the size of vocabulary and the size of the corpus.

## Document Parsing

Document parsing is to represent the content and structure of text documents. The primary content of most documents is the words.

Recognizing each word occurrence in a text document is called tokenizing or lexical analysis. Apart from these words, there may be other types of content, such as metadata, images, graphics, code, tables or hyperlinks; where metadata is information about a document that is not part of the text content. Metadata content includes document attributes such as date and author, and, most importantly, the tags that are used by markup languages, such as HTML or XML.

The parser uses the tags and other metadata recognized in the document to interpret the document's structure based on the syntax

of the markup language (syntactic analysis) and to produce a representation of the document that includes both the structure and content.

Text pre-processing usually including the following components:

- Tokenizing – is the process of forming words from the sequence of characters in a document.
- Stopping - words that have little meaning apart from other words.
- Stemming - captures the relationships between different variations of a word.
- Finding phrases or n-grams – they are useful in IR and text analysis.

**Question 2.** Which of the following is FALSE? And explain why it is FALSE.

(1) Stemming is a component of text processing that captures the relationships between different variations of a word.
(2) Stemming reduces the different forms of a word that occur because of inflection (e.g., plurals, tenses) or derivation (e.g., making a verb into a noun by adding the suffixation) to a common stem.
(3) In general, using a stemmer for search applications with English text produces a small but noticeable improvement in the quality of results.
(4) A dictionary-based stemmer uses a small program to decide whether two words are related, usually based on knowledge of word suffixes for a particular language.

**Question 3. (N-grams)**
Typically, *n*-grams are formed from overlapping sequences of words., i.e. move *n*-word "window" one word at a time in a document. For example, *bigrams are* 2 words sequences, and *trigrams* are 3 words sequences.

The definition of Tropical fish is described in the following document:

> **Tropical fish** are generally those **fish** found in aquatic **tropical** environments around the world, including both **freshwater** and **saltwater** species. Fishkeepers often keep **tropical fish** in **freshwater** and **saltwater aquariums**.

Please design a python program to print all bigrams and trigrams of the above document that contain at least one of the highlighted key words ('fish', 'tropical', 'freshwater', 'saltwater', 'aquariums').

## 2. Information Extraction

Information extraction is a language technology that focuses on extracting structure from text, e.g., Identifying noun phrases, titles, or even bolded text.

Most of the recent research in information extraction has been concerned with features that have specific semantic content, such as named entities, relationships, and events.

**Named entity**

A named entity is a word or sequence of words (such as people's names, company or organization names, locations, time and date expressions, quantities, and monetary values) that is used to refer to something of interest in a particular application,

the process of recognizing them and tagging them in text is sometimes called semantic annotation.

Example of tagging information

Fred Smith, who lives at 10 Water Street, Springfield, MA, is a long!time collector of tropical fish.

```
<p>
<PersonName>
     <GivenName> Fred </GivenName>
     <Sn> Smith </Sn>
</PersonName>, who lives at
<address>
     <Street >10 Water Street</Street>,
     <City>Springfield</City>,
     <State>MA</State></address>, is a long!time  collector of
<b>tropical fish.</b>
</p>
```

Two main approaches have been used to build named entity recognizers: **rule based** and **statistical**.

A rule-based recognizer uses one or more lexicons (lists of words and phrases) that categorize names, e.g., people's names (given names, family names). In many cases, however, rules or patterns are used to verify an entity name or to find new entities that are not in the lists. For example, new person names could be recognized by rules such as "<title> <name>", where <title> would include words such as "Prof.", "Mr.", and "CEO".

A statistical entity recognizer uses a probabilistic model of the words in and around an entity, e.g., Hidden Markov Model (HMM) approach.

## Hidden Markov Model

In nature language, words can have many different meanings, e.g., "Marathon" could be the name of a race or a location in Greece.

Human beings tell the meaning of a word based on **the context** of the word, meaning the words that surround it. For instance, "Boston Marathon", *Marathon* is preceded by *Boston*, the text is almost certainly describing a race.

The context of a word can be described by modeling **the generation of the sequence of words** in a text as a process with the Markov property, meaning that the next word in the sequence depends on only a small number of the previous words, where a **Markov Model** describes a process as a collection of states with transitions between them; and each of the transitions has an associated probability.

**Question 4. (Markov chain)**
Assume when John is sad today, which isn't very usual: he either goes for a run, gobbles down ice cream or takes a nap next day. The Markov Chain depicted in the following **state diagram** has 3 possible states: "Sleep", "Run", and "Ice Cream".

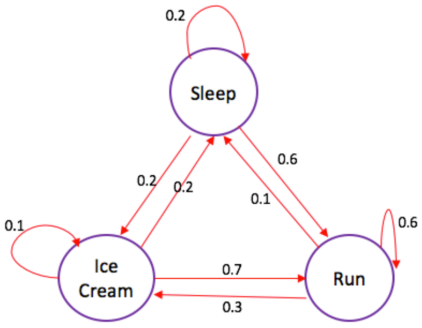


**Fig 2**. Markov chain example.

The following is the corresponding **transition matrix**

| CURRENT STATE \ NEXT STATE | SLEEP | RUN | ICE CREAM |
|---|---|---|---|
| SLEEP | 0.2 | 0.6 | 0.2 |
| RUN | 0.1 | 0.6 | 0.3 |
| ICE CREAM | 0.2 | 0.7 | 0.1 |

(1) According to the above diagram, if John spent sleeping a sad day away, what is the probability of he will likely go for a run next day?

(2) The transition matrix will be 3 × 3 matrix. To simple represent the matrix, we can use the following variables, for example,

SS – Sleep → Sleep
SR – Sleep → Run
SI – Sleep → Ice Cream
…

Then the transition matrix for the states is represented as a list of lists in python
[["SS","SR","SI"], ["RS","RR","RI"], ["IS","IR","II"]]

Please write the corresponding transition matrix for probabilities in a list of lists.

(3) (**Optional**) Design a function to forecast the state after *k* days. For example, you may assume the start state is 'Sleep'. You can also the "numpy.random.choice" to generate a random sample from the set of transitions for each day. You may have the following sample outputs:

Start state: Sleep
Possible states: ['Sleep', 'Run', 'Run']
End state after 2 days: Run
Probability of the possible sequence of states: 0.3

Although **Markov** models can be used to generate new sentences, they are more commonly used to **recognize entities in a sentence.**

For example, for the following sentence:

```
Fred Smith, who lives at 10 Water Street, Springfield, MA, is a
long!time  collector of tropical fish.
```

Based on training data, the recognizer may find the sequence of states:

```
<start><person><not-an-entity><location><not-an-entity><end>
```

to have the highest probability for that model.

Then, the words that were associated with the entity categories in this sequence would then be tagged.

The key aspect of the above approach to entity recognition is that the probabilities in the sentence model must be estimated from training data. We can generate training data that consists of text manually annotated with the correct entity tags.

From this training data, we can work out the probability of words associated with a given category (i.e., output probabilities), and the probability of transitions between categories.

Please note we can only observe a sentence. The underlying states (entity categories, e.g., "location", or "person") are **hidden**. Fig 3 shows a **Hidden Markov Model (diagram)**. It assumes a patient has two states: "Healthy" and "Fever". A doctor cannot see the (hidden) states directly; but the patient can tell the doctor that she/he is "normal", "cold", or "dizzy" (the observations).

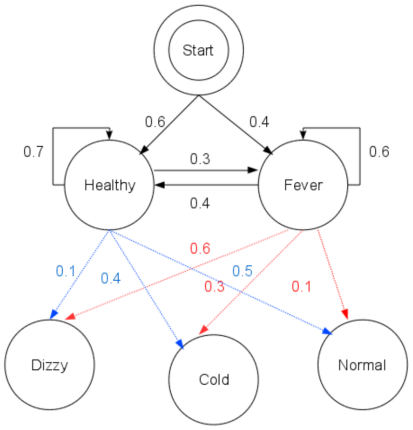


**Fig 3**. An example of **Hidden Markov Model**

The *observations* (e.g., normal, cold, dizzy) along with a *hidden* state (e.g., healthy, fever) form a hidden Markov model (HMM).

More formally, we have the following inputs and the output for finding the most likely sequence of states in an HMM.

**Input**

- The observation space $O = \{o_1, o_2, \ldots, o_N\}$,
- the state space $S = \{s_1, s_2, \ldots, s_K\}$,
- an array of initial probabilities $\Pi = (\pi_1, \pi_2, \ldots, \pi_K)$ such that $\pi_i$ stores the probability that $x_1 = s_i$,
- a sequence of observations $Y = (y_1, y_2, \ldots, y_T)$ such that $y_t = o_i$ if the observation at time $t$ is $o_i$,
- transition matrix $A$ of size $K \times K$ such that $A_{ij}$ stores the transition probability of transiting from state $s_i$ to state $s_j$,
- emission matrix $B$ of size $K \times N$ such that $B_{ij}$ stores the probability of observing $o_j$ from state $s_i$.

**Output**

- The most likely hidden state sequence $X = (x_1, x_2, \ldots, x_T)$

## **Viterbi algorithm** (this part is optional)

It is a dynamic programming algorithm. It can be used to find the most likely sequence of states in an HMM. The algorithm is described as follows:

**function** *VITERBI* $(O, S, \Pi, Y, A, B) : X$
  **for each state** $i = 1, 2, \ldots, K$ **do**
    $T_1[i, 1] \leftarrow \pi_i \cdot B_{iy_1}$
    $T_2[i, 1] \leftarrow 0$
  **end for**
  **for each observation** $j = 2, 3, \ldots, T$ **do**
    **for each state** $i = 1, 2, \ldots, K$ **do**
      $T_1[i, j] \leftarrow \max_k (T_1[k, j-1] \cdot A_{ki} \cdot B_{iy_j})$
      $T_2[i, j] \leftarrow \arg\max_k (T_1[k, j-1] \cdot A_{ki} \cdot B_{iy_j})$
    **end for**
  **end for**
  $z_T \leftarrow \arg\max_k (T_1[k, T])$
  $x_T \leftarrow s_{z_T}$
  **for** $j = T, T-1, \ldots, 2$ **do**
    $z_{j-1} \leftarrow T_2[z_j, j]$
    $x_{j-1} \leftarrow s_{z_{j-1}}$
  **end for**
  **return** $X$
**end function**

The calculation of probabilities in VITERBI can be understood as the chain rule. For two random variables *x* and *y* to find the joint distribution, we can apply the definition of conditional probability to obtain:

$$P(x, y) = P(y|x) * P(x)$$

Let $Pr(i, y_1) = \pi_i * B_{iy1}$ be the probability that $y_1$ occurs from stage $i$, where $\pi_i$ is the initial probability that stage $i$ is true. It is obvious that $T_1[i,1] = Pr(i, y_1)$.

Let $Pr(k, i, y_j)$ be the probability that $y_j$ occurs from stage i after we observed $y_{j-1}$, and state $i$ is the transition from stage $k$, i.e., $Pr(k, i, y_j) = T_1[k,j-1] * A_{ki} * B_{iyj}$

That is $T_1[i, j] = \max_k(Pr(k, i, y_j))$. For example, $T_1[i, 2] = \max_{k \in [1..K]}(Pr(k, i, y_2))$. Fig 4 shows the different transition paths (for $k \in [1..K]$) to stage i for the observation $y_2$.

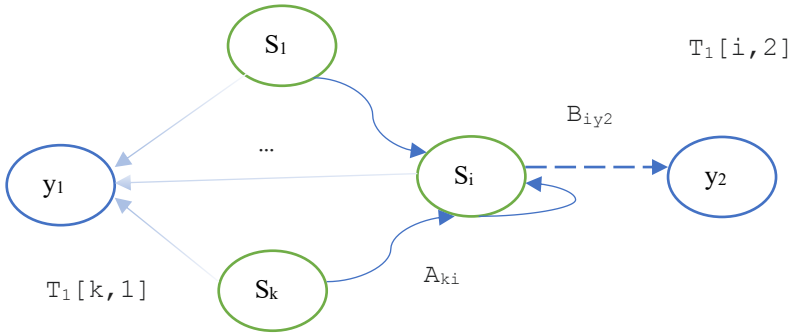


**Fig 4**. The different transition paths ($A_{ki}$) to stage i.

For the example in Fig 3, we can also represent the inputs as the following python definitions.

```
obs = ("normal", "cold", "dizzy") # assume Y = O here
states = ("Healthy", "Fever")
start_pi = {"Healthy": 0.6, "Fever": 0.4}
trans_A = {
    "Healthy": {"Healthy": 0.7, "Fever": 0.3},
    "Fever": {"Healthy": 0.4, "Fever": 0.6},
}
emit_B = {
    "Healthy": {"normal": 0.5, "cold": 0.4, "dizzy": 0.1},
    "Fever": {"normal": 0.1, "cold": 0.3, "dizzy": 0.6},
}
```

VITERBI Algorithm first initializes T1 and T2, where K=2, y1=normal, y2=cold, y3=dizzy; for the first observation y1, we have

```
T1[1,1] = 0.6*0.5 = 0.3
T1[2,1] = 0.4*0.1 = 0.04
T2[1,1] = 0
T2[2,1] = 0

For each rest observations:
For y2, j=2, i=1, we have

     T1[1,2] = max_{k=1,2}(0.3*0.7*0.4, 0.04*0.4*0.4)= 0.084
     T2[1,2] = 1 (k=1)


For y2, j=2, i=2, we have
     T1[2,2] = max_{k=1,2}(0.3*0.3*0.3, 0.04*0.6*0.3)= 0.027
     T2[2,2] = 1 (k=1)


For y3, j=3, i=1, we have
     T1[1,3] = max_{k=1,2}(0.084*0.7*0.1, 0.027*0.4*0.1)=
     0.00588
     T2[1,3] = 1 (k=1)


For y3, j=3, i=2, we have
     T1[2,3] = max_{k=1,2}(0.084*0.3*0.6, 0.027*0.6*0.6)=
     0.01512
     T2[2,3] = 1 (k=1)

T=3,

z_3 = 2
x_3 = fever

x_2 = health

x_1 = health
```

**Question 5. (<u>This question is optional</u>)**

For a given Hidden Markov Model, design a python Viterbi function. You can test your function by using Fig 3 and the above definition. For example, the following are the sample outputs when call it

viterbi(obs, states,start_pi,trans_A, emit_B)

```
         1       2       3
Healthy: 0.30000 0.08400 0.00588
```

```
Fever: 0.04000 0.02700 0.01512
The steps of states are Healthy Healthy Fever with highest probability of 0.01512
```

# 3. Document Structure and Markup

For web search, queries usually do not refer to document structure or fields, but that does not mean that document structure is unimportant. Some parts (e.g., the main heading for the page, and the anchor text for links) of the structure of web pages, indicated by HTML markup, are useful. The document parser must recognize this structure and make it available for indexing.

**Example:** HTML source for a Wikipedia page

## Tropical fish

**Tropical fish** include fish found in tropical environments around the world, including both freshwater and salt water species. Fishkeepers often use the term *tropical fish* to refer only those requiring fresh water, with saltwater tropical fish referred to as *marine fish*.

Tropical fish are popular aquarium fish , due to their often bright coloration. In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

https://en.wikipedia.org/wiki/Tropical_fish

```
<html>
<head>
<meta name="keywords" content="Tropical fish, Airstone, Albinism, Algae eater,
Aquarium, Aquarium fish feeder, Aquarium furniture, Aquascaping, Bath treatment
(fishkeeping),Berlin Method, Biotope" />
<title>Tropical fish - Wikipedia, the free encyclopedia</title>
</head>
<body>
<h1 class="firstHeading">Tropical fish</h1>
<p><b>Tropical fish</b> include <a href="/wiki/Fish" title="Fish">fish</a> found in <a
href="/wiki/Tropics" title="Tropics">tropical</a> environments around the world,
including both <a href="/wiki/Fresh_water" title="Fresh water">freshwater</a> and <a
href="/wiki/Sea_water" title="Sea water">salt water</a> species. <a
href="/wiki/Fishkeeping" title="Fishkeeping">Fishkeepers</a> often use the term
<i>tropical fish</i> to refer only those requiring fresh water, with saltwater tropical fish
referred to as <i><a href="/wiki/List_of_marine_aquarium_fish_species" title="List of
marine aquarium fish species">marine fish</a></i>.</p>
<p>Tropical fish are popular <a href="/wiki/Aquarium" title="Aquarium">aquarium</a>
fish , due to their often bright coloration. In freshwater fish, this coloration typically
derives from <a href="/wiki/Iridescence" title="Iridescence">iridescence</a>, while salt
water fish are generally <a href="/wiki/Pigment" title="Pigment">pigmented</a>.</p>
</body>
</html>
```

The above HTML source shows more structure, where each field or element in HTML is indicated by a start tag (such as **<h1>**) and an optional end tag (e.g., **</h1>**).

Elements can have attributes (with values), given by attribute_name = ”value” pairs. The **<head>** element contains metadata that is not displayed by a browser. The metadata element for keywords (<meta name = ”keywords”) gives a list of words and phrases that can be used as additional content terms for the page. The <title> metadata element gives the title for the page (which is different from the main heading).

The <body> element of the document contains the content that is displayed.

The main heading is indicated by the <h1> tag. Other headings, of different sizes and potentially different importance, would be indicated by <h2> through <h6> tags. Terms that should be displayed in bold or italic are indicated by <b> and <i> tags.

Links, such as <a href=”/wiki/Fish” title=”Fish”>fish</a>, are very common. They are the basis of link analysis algorithms such as PageRank, but also define the anchor text. Links and anchor text are of particular importance to web search.

**Question 6.**

Design a python program to extract all hyperlinks (or destination links) in a html file. You can use HTMLParser python package. You may download a real html file to test it.