3 Language Modelling

Unit III

Syllabus

Probabilistic language modeling, Markov models, Generative models of language, Log-Liner Models, Graph-based Models

N-gram models : Simple n-gram models, Estimation parameters and smoothing, Evaluating language models, Word Embeddings/ Vector Semantics: Bag-of-words, TFIDF, word2vec, doc2vec, Contextualized representations (BERT)

Topic Modelling : Latent Dirichlet Allocation (LDA), Latent Semantic Analysis, Non Negative Matrix Factorization

3.1 Probabilistic Language Modelling

- A language model in NLP is a probabilistic statistical model that determines the probability of a given sequence of words occurring in a sentence based on the previous words. It helps to predict which word is more likely to appear next in the sentence.
- Hence it is widely used in predictive text input systems, speech recognition, machine translation, spelling correction etc. The input to a language model is usually a training set of example sentences. The output is a probability distribution over sequences of words. We can use the last one word (unigram), last two words (bigram), last three words (trigram) or last n words (n-gram) to predict the next word as per our requirements.
- A language model is a distribution over sentences. For example, Consider the sentence :
  - John loves Mary more likely than the string Mary Mary John?
  - Which word is more likely to be next : John loves . ..
- Language models have a wide variety of uses :
  - In Speech Recognition (predict which word is going to be spoken).
  - In Mobile Phones (predict which letter is going to be typed)
  - Machine translation (order good translations from bad ones).
  - In theories of human language learning.
- Basic idea is to use probability theory to represent and process uncertainty. In probabilistic reasoning, the truth value of a proposition is extended from {0, 1} to [0, 1], with binary logic as its special case.
- **Justification :** though no conclusion is absolutely true, the one with the highest probability is preferred. Under certain assumptions, probability theory gives the optimum solutions.
- To extend the basic Boolean connectives to probability functions :
  - *Negation* : $P(\neg A) = 1 - P(A)$
  - *Conjunction* : $P(A \wedge B) = P(A) \times P(B)$ if A and B are independent of each other.

- *Disjunction* : P(AvB) = P(A) + P(B) if A and B never happen at the same time

- Furthermore, the conditional probability of B given A is $P(B|A) = \frac{P(B \wedge A)}{P(A)}$, from which Bayes' Theorem is derived, and it is often used to update a system's belief according to new information: $P(H|E) = P(E|H) \times \frac{P(H)}{P(E)}$

- **Bayesian Networks** are directed acyclic graphs in which the nodes represent variables of interest and the links represent informational or causal dependencies among the variables. The strength of dependency is represented by conditional probabilities. Compared to other approaches of probabilistic reasoning, Bayesian network is more efficient, though its actual computational cost is still high for complicated problems.

- Challenges to probabilistic approaches :
  - Unknown probability values
  - Inconsistent probability assignments
  - Computational expense

1. Inference using full joint distributions :

- **Probabilistic Inference :** Probabilistic Inference is the computation of posterior probabilities for query propositions given observed evidence. The full joint distribution is used as the "knowledge base" from which answers to all questions may be derived. Along the way we also introduce several useful techniques for manipulating equations involving probabilities.

Table 3.1.1 : A full joint distribution for the Toothache, Cavity, Catch world

	Toothache		¬ Toothache	
	Catch	¬ Catch	Catch	¬ Catch
Cavity	0.108	0.012	0.072	0.008
¬ Cavity	0.016	0.064	0.144	0.576

- As given in table a domain consisting of just the three Boolean variables Toothache, Cavity, and Catch. The full joint distribution is a 2 × 2 × 2 table.

- The probability associated with a proposition is defined to be the sum of the probabilities of the worlds in which it holds :

$$\text{For any proposition } \phi,\ P(\phi) = \sum_{\omega \in \phi} P(\omega) \qquad \ldots (3.1.1)$$

- Notice that the probabilities in the joint distribution sum to 1, as required by the axioms of probability. Notice also that Equation (3.1.1) gives us a direct way to calculate the probability of any proposition, simple or complex: simply identify those possible worlds in which the proposition is true and add up their probabilities. For example, there are six possible worlds in which cavity v toothache holds :

$$P(\text{cavity} \vee \text{toothache}) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.0064 = 0.28$$

- One typical job is to compute the distribution across a selection of variables or a single variable. Adding the entries in the first row, for example, yields the unconditional or marginal probability of cavity :

$$P(\text{cavity}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$$

- This process is called marginalization, or summing out. It is called marginalization because we sum up the probabilities for each possible value of the other variables, thereby taking them out of the equation. For each collection of variables Y and Z, we may construct the following generic marginalisation rule :

$$P(Y) = \sum_{z \in Z} P(Y, z) \quad \ldots (3.1.2)$$

Where $\sum_{z \in Z}$ means to sum over all the probable combinations of values of the set of variables Z. sometimes it is abbreviated as $\sum_{z}$, leaving Z implicit. We simply followed the rule.

$$P(\text{cavity}) = \sum_{z \in \{\text{Catch, Toothache}\}} P(\text{cavity}, z)$$

- **Conditioning** A variation of this approach uses conditional probabilities rather than joint probabilities and employs the product rule :

$$P(Y) = \sum_{z} P(Y \mid z)\, P(z) \quad \ldots (3.1.3)$$

- So, this rule is known as conditioning. Marginalization and conditioning prove to be useful rules for all types of probability expression derivations.
- In most circumstances, we want to compute conditional probabilities for certain variables given evidence for others. To get conditional probabilities, first use Equation $P(a|b) = \frac{p(a \wedge b)}{p(b)}$, to create an expression in terms of unconditional probabilities, and then evaluate the expression from the whole joint distribution. For example, given evidence of a toothache, we may compute the likelihood of a cavity as follows :

$$P(\text{cavity} \mid \text{toothache}) = \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})}$$

$$= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6$$

- To double-check, given a toothache, we may compute the likelihood that there is no cavity :

$$P(\neg \text{ cavity} \mid \text{toothache}) = \frac{P(\neg \text{ cavity} \wedge \text{toothache})}{P(\text{toothache})}$$

$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

- **Normalization :** As expected, in the above likelihood computation the two values add up to 1.0. Take note that no matter what value of Cavity we calculate, the term 1/P(toothache) remains constant in these two equations. In fact, it may be thought of as a normalisation constant for the distribution P (Cavity |toothache), guaranteeing that it adds up to 1. We may express the two preceding equations in one using this notation.

$$P(\text{Cavity} \mid \text{toothache}) = \alpha\, P(\text{Cavity, toothache})$$

$$= \alpha\, [P(\text{Cavity, toothache, catch}) + P(\text{Cavity, toothache, } \neg \text{ catch})]$$

$$= \alpha\, [\langle 0.108, 0.016\rangle + \langle 0.012, 0.064\rangle] = \alpha\, \langle 0.12, 0.08\rangle = \langle 0.6, 0.4\rangle$$

- In other words, even if we don't know the value of P(toothache), we can compute P(Cavity |toothache)! We temporarily disregard the component 1/P(toothache) and sum the values for cavity and cavity, yielding 0.12 and 0.08 respectively. These are the proper relative proportions, but they don't add up to one, so we normalise them by dividing each by 0.12 + 0.08 to get the real probabilities of 0.6 and 0.4.

- Normalization proves to be a handy shortcut in many probability calculations, both for making the computation quicker and for allowing us to proceed when some probability assessment (such as P(toothache)) is unavailable.
- We may derive a generic inference technique from the case. We begin with the instance when the query only has one variable, X. (Cavity in the example). Let E be the list of evidence variables (in this case, merely Toothache), e be the list of observed values for them, and Y be the remaining unobserved variables (just Catch in the example). P(X | e) is the question, and it may be evaluated as :

$$P(X \mid e) = \alpha P(X, e) = \alpha \sum_{y} P(X, e, y) \quad \ldots (3.1.4)$$

where the total is calculated across all potential ys (i.e., all possible combinations of values of the unobserved variables Y). Because the variables X, E, and Y together comprise the entire set of variables for the domain, P(X, e, y) is essentially a subset of probabilities from the full joint distribution.

- Equation (3.1.4) can answer probabilistic questions for discrete variables when given the whole joint distribution to deal with. However, it does not scale well : given a domain characterised by n Boolean variables, it requires an input table of size O(2n) and processes the table in O(2n) time.
- In a real-world issue, n might easily exceed 100, rendering O(2n) unworkable. The complete joint distribution in tabular form is just not a useful tool for developing reasoning systems. Instead, it should be seen as the theoretical framework upon which more successful techniques might be developed, in the same way that truth tables created the theoretical groundwork for more practical algorithms such as DPLL.

Ex. 3.1.1 : In an examination, 80% of the students passed in English, 85% in Mathematics and 75% in both English and Mathematics. If 40 students failed in both the subjects, find the total number of students.

Soln. :

Let the total number of student is x

Number of students passed in both subject is

$$n(A \cup B) = n(A) + n(B) - (A \cap B)$$

Here, $n(A) = 80\ \%$ of x

$n(B) = 85\ \%$ of x

$n(c) = 75\%$ of x

Therefore,

$$(A \cup B) = \frac{80}{100}X + \frac{85}{100}X + \frac{75}{100}X$$

$$(A \cup B) = \frac{90}{100}X = \frac{9}{100}X$$

Failed in both subjects

$$x - \frac{9x}{10} = \frac{x}{10}$$

$$\frac{x}{10} = 40$$

$$x = 400$$

Ex. 3.1.2 : In a class, there are 70% of students who prefer English and 40% of students who like both English and mathematics; what is the percentage of students who like both English and mathematics ?

Soln. :

let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\ \%$$

3.1.1 Bayes Theorem

- Reverend Thomas Bayes is remembered through Bayes' Theorem or Bayes' Rule. It depicts the likelihood of an occurrence based on past knowledge of factors that may be associated with that event. It may also be used as an example of conditional probability.
- As an example : There are three bags, each with some white marbles and some black marbles. If a random white marble is selected. It's likely that this white marble came from the first bag. In such instances, we employ the Bayes' Theorem. It is employed when the chance of an event occurring is computed dependent on other factors, which is also known as conditional probability.
- The most significant rule in Artificial Intelligence is Bayes' Rule. Given some evidence, it is the mathematical rule that specifies how to update a belief. To put it another way, it defines the act of learning.

  The equation itself is not too complex :

$$P(A \mid B) = \frac{P(A) \times P(B \mid A)}{P(B)}$$

$$\underbrace{\boxed{P(A|B)}}_{\text{posterior}} = \underbrace{\boxed{P(A)}}_{\text{prior}} \times \frac{\boxed{P(B|A)}\ \text{likelihood}}{\boxed{P(B)}\ \text{marginal}}$$

The equation : Posterior = Prior × (Likelihood over Marginal probability)

- There are four parts :
  - **Posterior probability** is the updated probability after the evidence is considered
  - **Prior probability** is the probability before the evidence is considered
  - **Likelihood** is the probability of the evidence, given the belief is true
  - **Marginal probability** is the probability of the evidence, under any circumstance
- Bayes' Rule can answer a variety of probability questions, which help us (and machines) understand the complex world we live in.

3.1.2 Conditional Probability

- Conditional probability is the first notion to grasp. You may already be familiar with the concept of probability in general. It enables you to think about unpredictable occurrences with the accuracy and rigour associated with mathematics.
- Conditional probability serves as a link that allows you to discuss how several uncertain occurrences are connected. It allows you to discuss how the likelihood of an occurrence might change depending on the circumstances.

- Consider the chance of winning a race on the assumption that you did not sleep the night before. You could anticipate this probability to be lower than the likelihood of winning if you'd gotten a full night's sleep.

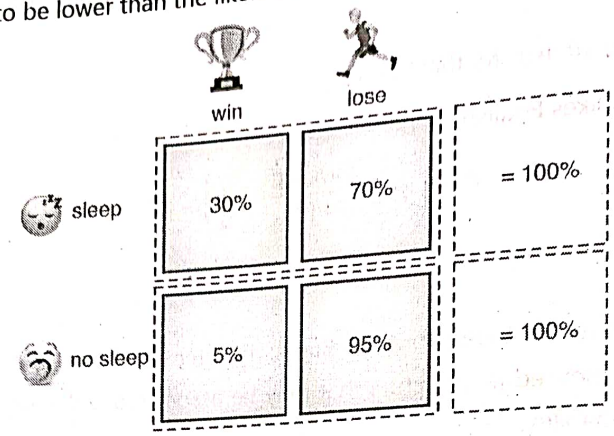


Fig. 3.1.1

- Consider the likelihood that a suspect committed a crime if their fingerprints are recovered at the scene. If their fingerprints were not recovered, the likelihood that they are guilty would be higher.
- The notation for conditional probability is usually :

  P(A|B)

- Which is read as "the probability of event A occurring, given event B occurs".
- That example, "probability of event A given event B" does not equal "probability of event B given event A."Consider the following example to help you remember:
- The likelihood of clouds provided that it is raining (100%) is not the same as the chance of rain given that there are clouds.
- Bayes' Rule tells you how to calculate a conditional probability with information you already have.
- It is helpful to think in terms of two events – a hypothesis (which can be true or false) and evidence (which can be present or absent).
- However, it can be applied to any type of events, with any number of discrete or continuous outcomes.

$$P(\text{Hypothesis} \mid \text{Evidence}) = P(\text{Hypothesis}) \times \frac{P(\text{Evidence} \mid \text{Hypothesis})}{P(\text{Evidence})}$$

- Bayes' Rule lets you calculate the posterior (or "updated") probability. This is a conditional probability. It is the probability of the hypothesis being true, if the evidence is present.
- Think of the prior (or "previous") probability as your belief in the hypothesis before seeing the new evidence. If you had a strong belief in the hypothesis already, the prior probability will be large.
- The prior is multiplied by a fraction. Think of this as the "strength" of the evidence. The posterior probability is greater when the top part (numerator) is big, and the bottom part (denominator) is small.
- The numerator is the **likelihood**. This is another conditional probability. It is the probability of the evidence being present, given the hypothesis is true.
- This is not the same as the posterior! Remember that the "probability of the hypothesis being true provided the evidence is present" is not the same as the "probability of the hypothesis being true given the evidence is present. "Take a peek at the denominator now. This is the evidence's marginal probability. That is, whether the hypothesis is true or wrong, it is the likelihood of the evidence being there. The evidence is more "convincing" when the denominator is small.

Ex. 3.1.3 : A doctor is aware that illness meningitis creates a stiff neck in 80 percent of patients. He is also aware of the following facts:

The known likelihood of a patient having meningitis illness is 1/30,000.

The known chance that a patient has a stiff neck is 2%.

Let a be the claim that the patient has a stiff neck and b be the statement that the patient has meningitis.

Soln. :

so we can calculate the following as :

$$P(A|B) = 0.8$$

$$P(B) = \frac{1}{30000}$$

$$P(A) = .02$$

$$P(A \mid B) = \frac{P(A) \times P(B \mid A)}{P(B)} = \frac{08 \times \left(\frac{1}{3000}\right)}{0.02} = 0.00133333$$

Ex. 3.1.4 : Spam Assassin functions by allowing users to train the system. It searches for patterns in the text in emails that the user has flagged as spam. For example, it may have discovered that the term "free" appeared in 20% of spam emails. Assuming that 0.1 percent of non-spam mail contains the term "free" and that 50% of all mails received by the user are spam, calculate the likelihood that a letter contains the phrase "free."

Soln. :

- P(Free | Spam) = 0.20
- P(Free | Non Spam) = 0.001
- P(Spam) = 0.50 $\Rightarrow$ P(Non Spam) = 0.50
- P(Spam | Free) = ?

Using Bayes' Theorem :

$$P(A \mid B) = \frac{P(A) \times P(B \mid A)}{P(B)}$$

$$P(\text{Spam} \mid \text{Free}) = \frac{P(\text{Spam}) \times P(\text{Free} \mid \text{Spam})}{P(\text{Free})}$$

$$P(\text{Spam} \mid \text{Free}) = \frac{0.50 \times 0.20}{0.50 \times 0.20 + 0.50 \times 0.001}$$

$$P(\text{Spam} \mid \text{Free}) = 0.995$$

Ex. 3.1.5 : There are three boxes labelled A, B, and C. The following are the box specifics :

Box A has two red and three black balls.

Box B has three red and one black ball.

Box C has one red ball and four black balls.

All three boxes are similar and have an equal chance of being picked up. So, what is the likelihood that the red ball was picked up from box A ?

Soln. : Let E denote the event that a red ball is picked up and A, B and C denote that the ball is picked up from their respective boxes.

Therefore, the conditional probability would be P(A|E) which needs to be calculated.

The existing probabilities $P(A) = P(B) = P(C) = \frac{1}{3}$,

Since all boxes have equal probability of getting picked.

$$P(E|A) = \frac{\text{Number of red balls in box A}}{\text{Total number of balls in box A}} = \frac{2}{5}$$

Similarly, $P(E|B) = \frac{3}{4}$

$$P(E|C) = \frac{1}{5}$$

Then evidence $P(E) = P(E|A) \times P(A) + P(E|B) \times P(B) + P(E|C) \times P(C)$

$$= \left(\frac{2}{5}\right) \times \left(\frac{1}{3}\right) + \left(\frac{3}{4}\right) \times \left(\frac{1}{3}\right) + \left(\frac{1}{5}\right) \times \left(\frac{1}{3}\right) = 0.45$$

Therefore, $P(A|E) = \frac{P(E|A) \times P(A)}{P(E)} = \frac{(2/5) \times (1/3)}{0.45} = 0.296$

3.2 Markov Models

- The Hidden Markov model is a probabilistic model which is used to explain or derive the probabilistic characteristic of any random process. It basically says that an observed event will not be corresponding to its step-by-step status but related to a set of probability distributions.
- Let's assume a system that is being modelled is assumed to be a Markov chain and in the process, there are some hidden states. In that case, we can say that hidden states are a process that depends on the main Markov process/chain.
- The main goal of HMM is to learn about a Markov chain by observing its hidden states. Considering a Markov process X with hidden states Y here the HMM solidifies that for each time stamp the probability distribution of Y must not depend on the history of X according to that time.
- To explain it more we can take the example of two friends, Rahul and Ashok. Now Rahul completes his daily life works according to the weather conditions. Major three activities completed by Rahul are- go jogging, go to the office, and cleaning his residence.
- What Rahul is doing today depends on whether and whatever Rahul does he tells Ashok and Ashok has no proper information about the weather But Ashok can assume the weather condition according to Rahul work.
- Ashok believes that the weather operates as a discrete Markov chain, wherein the chain there are only two states whether the weather is Rainy or it is sunny. The condition of the weather cannot be observed by Ashok, here the conditions of the weather are hidden from Ashok.
- On each day, there is a certain chance that Bob will perform one activity from the set of the following activities {"jog", "work"," clean"}, which are depending on the weather. Since Rahul tells Ashok that what he has done, those are the observations. The entire system is that of a Hidden Markov Model (HMM).

- Here we can say that the parameter of HMM is known to Ashok because he has general information about the weather and he also knows what Rahul likes to do on average.
- So let's consider a day where Rahul called Ashok and told him that he has cleaned his residence. In that scenario, Ashok will have a belief that there are more chances of a rainy day and we can say that belief Ashok has is the start probability of HMM let's say which is like the following :
- The states and observation are :

states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')

- And the start probability is :

start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

- Now the distribution of the probability has the weightage more on the rainy day stateside so we can say there will be more chances for a day to being rainy again and the probabilities for next day weather states are as following

```
transition_probability = {
'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
}
```

- From the above we can say the changes in the probability for a day is transition probabilities and according to the transition probability the emitted results for the probability of work that Rahul will perform is

```
emission_probability = {
'Rainy' : {'jog': 0.1, 'work': 0.4, 'clean': 0.5},
'Sunny' : {'jog': 0.6, 'work: 0.3, 'clean': 0.1},
}
```

- This probability can be considered as the emission probability. Using the emission probability Ashok can predict the states of the weather or using the transition probabilities Ashok can predict the work which Rahul is going to perform the next day. Fig. 3.2.1 shown the HMM process for making probabilities

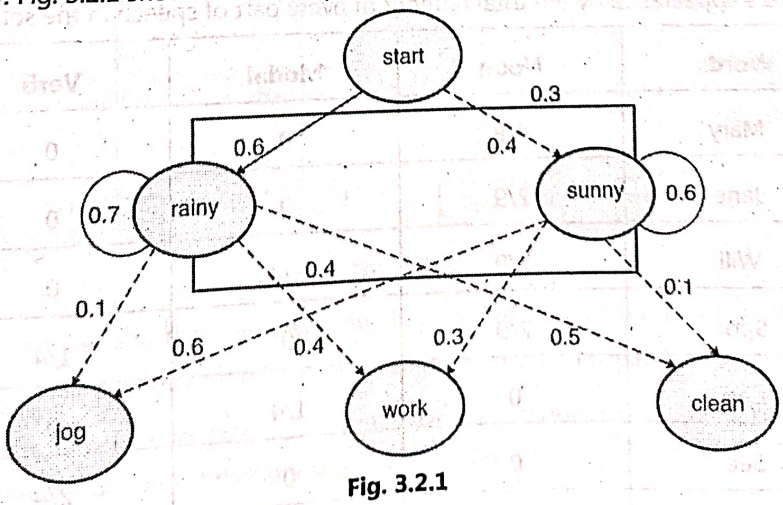


Fig. 3.2.1

- So here from the above intuition and the example we can understand how we can use this probabilistic model to make a prediction. Now let's just discuss the applications where it can be used.

POS Tagging With Hidden Markov Model :

- We can say that in the case of HMM is a stochastic technique for POS tagging. Let's take an example to make it more clear how HMM helps in selecting an accurate POS tag for a sentence.
- As we have seen in the example of the HMM process in POS tagging the transition probability is the likelihood of any sequence for example what are the chances for a noun word to come after any modal and a modal after a verb and a verb after a noun.
- Let's take the sentence "Rahul will eat food" where Rahul is a noun, will is a modal, eat is a verb and food is also a noun, so the probability for a word to be in a particular class of part of speech is called the Emission probability.
- Let's take a look at how we can calculate these two probabilities for a set of sentences:
  - Mary Jane can see will
  - The spot will see Mary
  - Will Jane spot Mary?
  - Mary will pat Spot
- The below table is a counting tableau for the words with their part of speech type

Words	Noun	Modal	Verb
mary	4	0	0
jane	2	0	0
will	1	3	0
spot	2	0	1
can	0	1	0
see	0	0	2
pat	0	0	1

- Let's divide each word's appearance by the total number of every part of speech in the set of sentences.

Words	Noun	Modal	Verb
Mary	4/9	0	0
Jane	2/9	0	0
Will	1/9	3/4	0
Spot	2/9	0	1/4
Can	0	1/4	0
See	0	0	2/4
Pat	0	0	1/4

- Here in the table, we can see the emission probabilities of every word.
- Now as we have discussed that the transition probability is the probability of the sequences we can define a table for the above set of sentences according to the sequence of part of speech.

	Noun	Modal	Verb	End
Start	3	1	0	0
Noun	1	3	1	4
Model	1	0	3	0
Verb	4	0	0	0

- Now in the table, we are required to check for the combination of parts of speeches for calculation of the transition probabilities. For example, we can see in the set of sentences modal before a verb has appeared 3 times and 1 time before a noun. This means it has appeared in the set for four-time and the probability of coming modal before any verb will be 3/4 and before a noun will be 1/4 Similarly performing this for every entity of the table :

	Noun	Modal	Verb	End
Start	3/4	1/4	0	0
Noun	1/9	3/9	1/9	4/9
Model	1/4	0	3/4	0
Verb	4/4	0	0	0

- Here the above values in the table are the respective transition values for a given set of sentences.
- Let's take the sentence "Will Jane spot Mary?" out from the set and now we can calculate the probabilities for every part of speech using the above calculations.

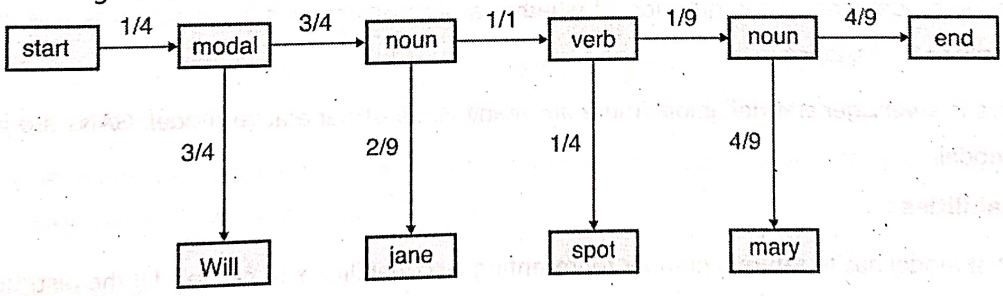


Fig. 3.2.2

- In the above image, we can see that we have emission probabilities of the words in the sentence given in the vertical lines and the horizontal lines are representing all the transition probabilities.
- Now the correctness of the POS tagging is measured by the product of all these probabilities. The product of probabilities represents the likelihood that the sequence is right.
- Let's just check for the rightness of the POS tagging.

$$\frac{1}{4}\times\frac{3}{4}\times\frac{3}{4}\times\frac{2}{9}\times\frac{1}{1}\times\frac{1}{4}\times\frac{1}{9}\times\frac{4}{9}\times\frac{4}{9} = 0.0001714678$$

- Here we can see that the product is higher than zero which means the POS tagging we have performed is correct and if the result is zero then the tagging performed will not be correct.
- So here we have seen how the HMM,s algorithm works for providing the POS tagging to the sentences but the example was small where we had only 3 kinds of POS tags but there are 81 different kinds of POS tags available.
- When it comes to finding the number of combinations from a small data set it can be done with smaller efforts but when it comes to tagging larger sentences and finding the right sequences with all the 81 tags the number of combinations increases exponentially. The computation may cause a larger effect but the more number of POS tags gives more accuracy.

3.3 Generative Models of Language

- What does "generative" mean in the name "Generative Adversarial Network"? "*Generative*" describes a class of statistical models that contrasts with *discriminative* models.
- Informally :
  - **Generative** : Models can generate new data instances.
  - **Discriminative** : Models discriminate between different kinds of data instances.
- A generative model could generate new photos of animals that look like real animals, while a discriminative model could tell a dog from a cat. GANs are just one kind of generative model.
- More formally, given a set of data instances X and a set of labels Y :
  - **Generative** models capture the joint probability p(X, Y), or just p(X) if there are no labels.
  - **Discriminative** models capture the conditional probability p(Y | X).
- A generative model includes the distribution of the data itself, and tells you how likely a given example is. For example, models that predict the next word in a sequence are typically generative models (usually much simpler than GANs) because they can assign a probability to a sequence of words.
- A discriminative model ignores the question of whether a given instance is likely, and just tells you how likely a label is to apply to the instance.
- Note that this is a very general definition. There are many kinds of generative model. GANs are just one kind of generative model.

Modeling Probabilities :

- Neither kind of model has to return a number representing a probability. You can model the distribution of data by imitating that distribution.
- For example, a discriminative classifier like a **decision tree** can label an instance without assigning a probability to that label.
- Such a classifier would still be a model because the distribution of all predicted labels would model the real distribution of labels in the data.
- Similarly, a generative model can model a distribution by producing convincing "fake" data that looks like it's drawn from that distribution.

Generative Models are Hard :

- Generative models tackle a more difficult task than analogous discriminative models. Generative models have to model more.
- A generative model for images might capture correlations like "things that look like boats are probably going to appear near things that look like water" and "eyes are unlikely to appear on foreheads." These are very complicated distributions.
- In contrast, a discriminative model might learn the difference between "sailboat" or "not sailboat" by just looking for a few tell-tale patterns. It could ignore many of the correlations that the generative model must get right.
- Discriminative models try to draw boundaries in the data space, while generative models try to model how data is placed throughout the space. For example, the following diagram shows discriminative and generative models of handwritten digits :

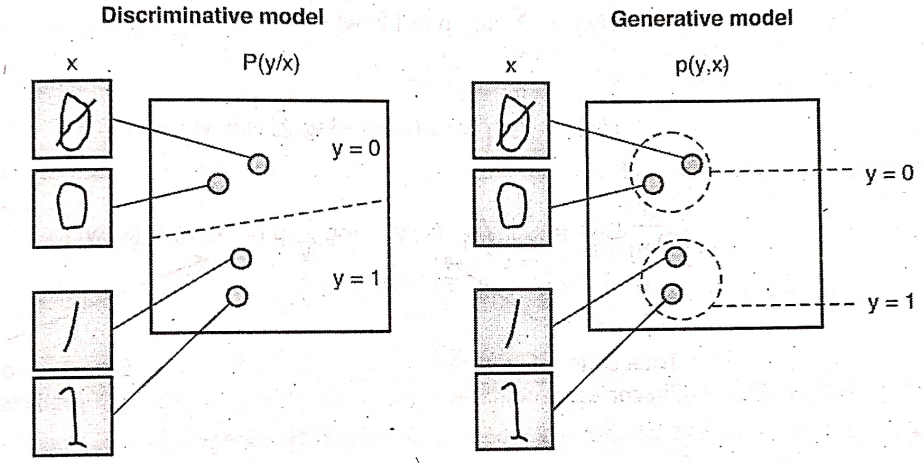


Fig. 3.3.1 : Discriminative and generative models of handwritten digits.

- The discriminative model tries to tell the difference between handwritten 0's and 1's by drawing a line in the data space. If it gets the line right, it can distinguish 0's from 1's without ever having to model exactly where the instances are placed in the data space on either side of the line.
- In contrast, the generative model tries to produce convincing 1's and 0's by generating digits that fall close to their real counterparts in the data space. It has to model the distribution throughout the data space.
- GANs offer an effective way to train such rich models to resemble a real distribution. To understand how they work we'll need to understand the basic structure of a GAN.

3.4 Log-Liner Models

- A **log-linear model** is a mathematical model that takes the form of a function whose logarithm equals a linear combination of the parameters of the model, which makes it possible to apply (possibly multivariate) linear regression. That is, it has the general form. Log-linear models have become a widely-used tool in NLP classification tasks. Log-linear models assign joint probabilities to observation / label pair $(x, y) \in X \times Y$.
- We have some input domain X, and a finite lable set y. Aim is to provide a conditional probability P (y | x) for any $x \in X$ and $y \in Y$.

- A feature is a function $f : X \times Y \rightarrow \mathbb{R}$ (Often binary features or indicator functions $f : X \times Y \rightarrow \{0, 1\}$)
- Say we have m features ϕ_k for $k = 1 \ldots m \Rightarrow$ A feature vector $\phi(x, y) \in \mathbb{R}^m$ for any $x \in X$ and $y \in Y$
- We also have parameter vector $W \in \mathbb{R}^m$

We define

$$p(y \mid x; w) = \frac{\exp(w \cdot \phi(x, y))}{\sum_{y'} \exp(w \cdot \phi(x, y'))}$$

Derivative of log linear model :

- Unfortunately, $\text{argmax}_w L(w)$ doesn't have a close formed solution
- We will have to differentiate and use gradient ascent

$$L(w) = \sum_{i=1}^{n} \log p(y_i \mid x_i; w)$$

$$L(w) = \sum_{i=1}^{n} \left(w \cdot \phi(x_i, y_i) - \log \sum_{y} \exp(w \cdot \phi(x_i, y)) \right)$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^{n} \left(\phi_j \cdot (x_i, y_i) - \log \sum_{y} p(y \mid x_i; w)\, \phi_j(x_i, y) \right)$$

Total count of feature j in correct candidates

Expected count of feature j in predicted candidates

Conditional Likelihood derivation :

- Recall

$$p(y \mid x; w) = \frac{\exp(w \cdot \phi(x, y))}{\sum_{y'} \exp(w \cdot \phi(x, y'))}$$

$$P(Y \mid X, w) = \prod_{(x, y) \in D} p(y \mid x, w)$$

- We can separate this into two components :

$$\log P(Y \mid X, w) = \sum_{(x, y) \in D} \log \exp \sum_{i} w_i \phi_i(x, y) - \sum_{(x, y) \in D} \log \sum_{y'} \exp \sum_{i} w_i \phi_i(x, y')$$

- The derivative is the difference between the derivatives of each component

$$\log P(Y \mid X, w) = N(w) - D(w)$$

3.5 Graph-based Models

- *Graphs are a general language for describing and analyzing entities with relations/interactions* Graphs are prevalent all around us from computer networks to social networks to disease pathways. Networks are often referred to as graphs that occur naturally, but the line is quite blurred and they do get used interchangeably.

- A knowledge graph is a way of storing data that resulted from an information extraction task. Many basic implementations of knowledge graphs make use of a concept we call triple, that is a set of three items(a subject, a predicate and an object) that we can use to store information about something.

Represent Knowledge in a Graph

- We can define a graph as a set of nodes and edges.

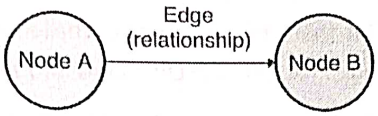


Fig. 3.5.1

- Node A and Node B here are two different entities. These nodes are connected by an edge that represents the relationship between the two nodes. Now, this is the smallest knowledge graph we can build – it is also known as a **triple**. Knowledge Graph's come in a variety of shapes and sizes.

How to Represent Knowledge in a Graph ?

- Before we get started with building Knowledge Graphs, it is important to understand how information or knowledge is embedded in these graphs.
- Let me explain this using an example. If Node A = Putin and Node B = Russia, then it is quite likely that the edge would be "president of":

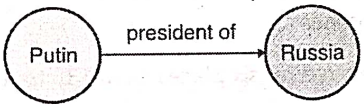


Fig. 3.5.2

- A node or an entity can have multiple relations as well. Putin is not only the President of Russia, he also worked for the Soviet Union's security agency, KGB. But how do we incorporate this new information about Putin in the knowledge graph above?
- It's actually pretty simple. Just add one more node for the new entity, KGB:

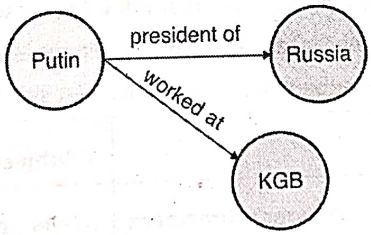


Fig. 3.5.3

- The new relationships can emerge not only from the first node but from any node in a knowledge graph as shown Fig. 3.5.4 :

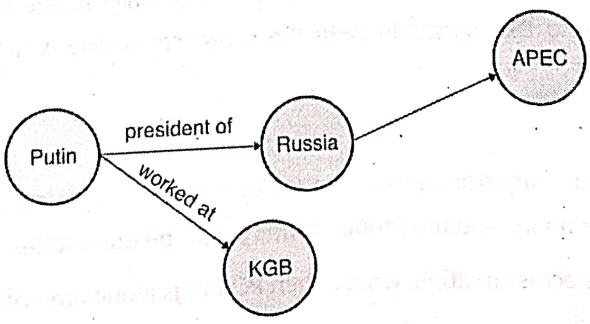


Fig. 3.5.4

- Russia is a member of the Asia Pacific Economic Cooperation (APEC).
- Identifying the entities and the relation between them is not a difficult task for us. However, manually building a knowledge graph is not scalable. Nobody is going to go through thousands of documents and extract all the entities and the relations between them!
- That's why machines are more suitable to perform this task as going through even hundreds or thousands of documents is child's play for them. But then there is another challenge – machines do not understand natural language. This is where Natural Language Processing (NLP) comes into the picture.
- To build a knowledge graph from the text, it is important to make our machine understand natural language. This can be done by using NLP techniques such as sentence segmentation, dependency parsing, parts of speech tagging, and entity recognition. Let's discuss these in a bit more detail.

Sentence Segmentation :

- **The first step in building a knowledge graph is to split the text document or article into sentences.** Then, we will shortlist only those sentences in which there is exactly 1 subject and 1 object. Let's look at a sample text below:
- "Indian tennis player Sumit Nagal moved up six places from 135 to a career-best 129 in the latest men's singles ranking. The 22-year-old recently won the ATP Challenger tournament. He made his Grand Slam debut against Federer in the 2019 US Open. Nagal won the first set."
- Let's split the paragraph above into sentences :
  1. Indian tennis player Sumit Nagal moved up six places from 135 to a career-best 129 in the latest men's singles ranking
  2. The 22-year-old recently won the ATP Challenger tournament
  3. He made his Grand Slam debut against Federer in the 2019 US Open
  4. Nagal won the first set
- Out of these four sentences, we will shortlist the second and the fourth sentences because each of them contains 1 subject and 1 object. In the second sentence, "22-year-old" is the subject and the object is "ATP Challenger tournament". In the fourth sentence, the subject is "Nagal" and "first set" is the object :

Sentence	Subject	Object
The 22-year old recently won ATP challenger tournament	22-year old	ATP challenger tournament
Nagal won the first set	Nagal	First set

- The challenge is to make your machine understand the text, especially in the cases of multi-word objects and subjects. For example, extracting the objects in both the sentences above is a bit tricky. Can you think of any method to solve this problem?

Entities Extraction :

- The extraction of a single word entity from a sentence is not a tough task. We can easily do this with the help of parts of speech (POS) tags. The nouns and the proper nouns would be our entities.
- However, when an entity spans across multiple words, then POS tags alone are not sufficient. We need to parse the dependency tree of the sentence.

he ... det
22-year ... amod
– ... punct
old ... nsubj
recently ... advmod
won ... ROOT
ATP ... compound
Challenger ... compound
tournament ... dobj
. ... punct

- The subject (*nsubj*) in this sentence as per the dependency parser is "old". That is not the desired entity. We wanted to extract "22-year-old" instead.
- The dependency tag of "22-year" is *amod* which means it is a modifier of "old". Hence, we should define a rule to extract such entities.
- The rule can be something like this extract the subject/object along with its modifiers and also extract the punctuation marks between them.
- But then look at the object (*dobj*) in the sentence. It is just "tournament" instead of "ATP Challenger tournament". Here, we don't have the modifiers but compound words.
- Compound words are those words that collectively form a new term with a different meaning. Therefore, we can update the above rule to **extract the subject/object along with its modifiers, compound words and also extract the punctuation marks between them.**
- In short, we will use dependency parsing to extract entities.

Extract Relations :

- Entity extraction is half the job done. To build a knowledge graph, we need edges to connect the nodes (entities) to one another. These edges are the relations between a pair of nodes.
- Let's go back to the example in the last section. We shortlisted a couple of sentences to build a knowledge graph :

Sentence
The 22-year old recently won ATP challenger tournament
Nagal won the first set

- Can you guess the relation between the subject and the object in these two sentences ?
- Both sentences have the same relation – "won". Let's see how these relations can be extracted. We will again use dependency parsing :

Nagal ... nsubj
won ... ROOT
the ... det
first ... amod
set ... dobj
. ... punct

- To extract the relation, we have to find the ROOT of the sentence (which is also the verb of the sentence). Hence, the relation extracted from this sentence would be "won".
- Finally, the knowledge graph from these two sentences will be like this :

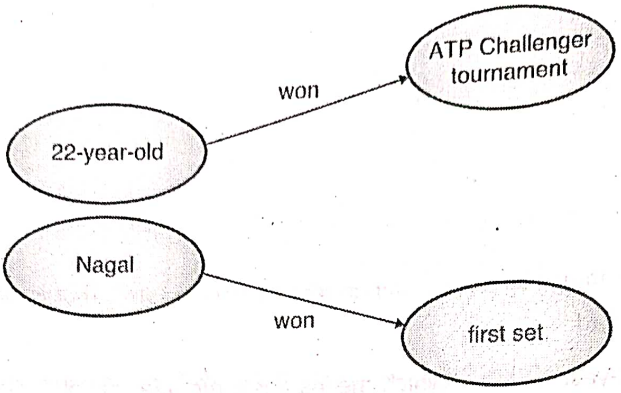


Fig. 3.5.6

3.6 N-gram Models

3.6.1 Simple n-gram Models

- An N-gram is a sequence of N words. Let's understand N-gram with an example. Consider the following sentence: "I love reading books about data science on Analytics Swati."
- A 1-gram/ unigram is a one-word sequence. For the given sentence, the unigrams would simply be: "I", "love", "reading", "books", "about", "data", "science", "on", "Analytics", "Swati".
- A 2-gram/bigram is a two-word sequence of words, such as "I love", "love reading", or "Analytics Swati".
- A 3-gram/ trigram is a three-word sequence of words like "I love reading", "about data science" or "on Analytics Swati".
- An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. If we have a good N-gram model, we can predict p(w | h) – what is the probability of seeing the word w given a history of previous words h – where the history contains n – 1 words.
- The estimation of probability of the sentence is achieved by decomposing sentence probability into a product of conditional probabilities. We compute this probability in two steps :
  1. Apply the chain rule of probability
  2. Apply a very robust simplification assumption to let us to compute $p(w_1 \ldots w_s)$ in an easy manner. apply chain of custody rule as follows :

$$P(s) = P(w_1 w_2 w_3 \ldots w_n)$$

$$= P(w_1) P\left(\frac{w_2}{w_1}\right) P\left(\frac{w_3}{w_1 w_2}\right) w_1 P\left(\frac{w_3}{w_1 w_2 w_3}\right) P\left(\frac{w_3}{w_1 w_2 w_3 \ldots w_n}\right)$$

$$= \pi_{i=1}^{n} P\left(\frac{w_i}{h_i}\right)$$

where h_i is history and is defined as $(w_1 w_2 w_3 \ldots w_{i-1})$

- So, to calculate the word probability we have to calculate the probability of the word, given the sequence of word preceding it.
- An n-gram model simplifies the task by approximating the probability of the word given all the previous words by the conditional probability given previous n – 1 word only.

$$P\left(\frac{w_i}{h_i}\right) = P\left(\frac{w_i}{w_{i-n+1}\, w_{i-1}}\right)$$

- Hence, N-gram model calculates $P\left(\frac{w_i}{h_i}\right)$ by modelling language as Markov model of order n – 1. It means by looking at the previous n – 1 word only. using the bigram and the trigram model the probability is estimated as :

$$P(s) = \pi_{i=1}^{n} P\left(\frac{w_i}{w_{i-1}}\right)$$

and trigram model,

$$P(s) = \pi_{i=1}^{n} P\left(\frac{w_i}{w_{i-1}\, w_{i-1}}\right)$$

3.6.2 Estimation Parameters and Smoothing

3.6.2(A) Estimation Parameters

1. Maximum Likelihood (ML) Estimation

- ML estimation tries to find the estimate of the parameter θ by maximizing the likelihood function.
- Assume we have i. i.d random samples $x_1, x_2, \ldots, x_n$ that follow a distribution $f(x_1, x_2, \ldots, x_n; \theta)$, which depends on the unknown parameter θ. The goal is to estimate this unknown quantity such that it maximizes the probability of observing this random sample. Following is how you formulate this problem using the maximum likelihood :

1. Construct the likelihood function $L(\theta|x) = f(x_1, x_2, \ldots, x_n; \theta)$
2. Use the property of i. i.d random samples to break the joint PDF to the product of n marginal PDF.

$$F(x_1, x_2, \ldots X_n; \theta) = \prod_{i=1}^{i=n} f(x_i; \theta)$$

3. Take a logarithm (usually a natural log) to change the product to summation. This does not change the optimal estimator since the logarithm is a monotonic function.

$$LL(\theta; x) = \sum_{i=1}^{i=n} f(x; \theta)$$

Where $LL(\theta; x)$ represents the log-likelihood function.

4. Differentiate the log-likelihood function with respect to θ and set it to zero.

$$\frac{\partial LL(\theta; x)}{\partial \theta} = 0$$

5. The estimator will only be a function of observed data.

$$\hat{\theta}_{ML} = g(x_1, x_2, \ldots, x_n)$$

2. Maximum a Posteriori (MAP) Estimation :

- MAP estimation tries to find the estimate of the parameter θ by maximizing the posterior distribution. Recall the Bayes law again but this time we are not trying to compute the exact value of posterior.
- This is important since we do not need to worry about the denominator because it is independent of the parameter. Therefore, all is needed is to maximize the product of likelihood and the prior probability.

$$P(\theta \mid x) = \frac{P(x|\theta)\, P(\theta)}{P(x)} \propto P(x \mid \theta)\, P(\theta)$$

- Remember, P(X|θ) is the same as the likelihood function. Therefore, the MAP estimate is the same as the ML estimate with the inclusion of the prior probability.
- Following is how you formulate the problem using the MAP :
  1. Construct the posterior distribution.
  2. Use the property of i.i.d random sample to simplify the posterior distribution.

$$P(\theta \mid x_1, x_2, \ldots, x_n) \propto P(x_1, x_2, \ldots, x_n \mid \theta)\, P(\theta) = P(\theta) \prod_{i=1}^{i=n} P(x_i \mid \theta)$$

Note : Slight abuse of notation to use letter P for the density function.

  3. Take the logarithm (Usually a natural log) to further simplify the above relationship.

$$\log P(\theta \mid x_1, x_2, \ldots, x_n) \propto \log P(\theta) + \sum_{i=1}^{i=n} P(x_i \mid \theta)$$

  4. Differentiate the above equation with respect to θ and set it to zero.

$$\frac{\partial \log P(\theta \mid x_1, x_2, \ldots, x_n)}{\partial \theta} = \frac{\partial \log P(\theta)}{\theta} + \frac{\partial \sum_{i=1}^{i=n} P(x_i \mid \theta)}{\partial \theta} = 0$$

Note : MAP and ML estimate is the same if θ follows the uniform distribution. What this means intuitively is that all values of θ have equal weight; therefore, knowing the distribution of θ does not give us any more useful distribution.

Ex. 3.6.1 : Consider a communication system that the transmitted signal $X \sim N(0,\sigma^2)$ (Gaussian distribution with zero mean and variance of σ_x^2). The received signal Y can be modeled as follows :

$Y = X + n$, Where $n \sim N(0,\sigma_n^2)$. We would like to find the MAP and ML estimates for transmitted signal X.

Soln. :

1. ML Estimation :

Y is a received or observed message. You can think of Y as a noisy version of X. This is a standard problem in the communication system. We never know what message is transmitted. (otherwise, there is no point in designing the receiver, error correction codes, etc.) To find the ML estimate, we need to follow the steps outlined above.

First, construct the likelihood function : P(Y|X=x). Then find the log-likelihood expression and then differentiate respect to x and set it to zero.

$$P(Y \mid X = x) = \frac{1}{\sqrt{2\pi}\,\sigma_n} e^{\frac{-(y-x)^2}{2\sigma_n^2}}$$

$$\ln P(Y \mid X = x) = \ln \frac{1}{\sqrt{2\pi}\,\sigma_n} - \frac{(y-x)^2}{2\sigma_n^2} = C_1 - \frac{(y-x)^2}{2\sigma_n^2}$$

C_1 denotes the constant terms that do not depend on x (We do not care about them since they are zero after differentiation).

$$\frac{\partial \ln P(Y \mid X = x)}{\partial x} = \frac{(y-x)}{\sigma_n^2} = 0$$

$$\hat{x}_{ML} = y$$

- **Interpretation :** The ML estimate of x is the observed message y. This means under the maximum likelihood estimation, the best estimate for the transmitted signal is the received noisy signal.
- **MAP Estimation :** First, we need to construct the posterior distribution by multiplying the likelihood and prior together (Remember, the denominator is not important since it is not a function of the parameter).

$$P(X|Y = y) \propto P(Y|X = x)\,P(X)$$

$$P(X|Y = y) \propto \frac{1}{\sqrt{2\pi}\,\sigma_n} e^{\frac{-(y-x)^2}{2\sigma_n^2}} \frac{1}{\sqrt{2\pi}\,\sigma_x} e^{\frac{-x^2}{2\sigma_x^2}} = \frac{1}{2\pi\,\sigma_n\,\sigma_x} e^{-\frac{\sigma_x^2 (y-x)^2 + \sigma_n^2 x^2}{2\sigma_x^2 \sigma_n^2}}$$

- The maximum value of the posterior occurs when the exponent is minimized. Differentiating respect to x and set it to zero will result in the following :

$$f(x) = \frac{\sigma_x^2 (y-x)^2 + \sigma_n^2 x^2}{2\sigma_x^2 \sigma_n^2}$$

$$\frac{\partial f(x)}{\partial x} = \frac{2\sigma_x^2 (y-x) + 2x\sigma_n^2}{2\sigma_n^2 \sigma_x^2} = 0$$

$$\hat{x}_{MAP} = \frac{\sigma_x^2}{\sigma_x^2 + \sigma_n^2} y$$

- **Interpretation :** The MAP estimate of x is linearly proportional to the received signal y. If the variance of signal is infinity (becomes very large), then the normal distribution becomes the uniform distribution, and the MAP and ML estimate becomes the same.

$$\lim_{\sigma_x^2 \to \infty} \hat{x}_{MAP} = \lim_{\sigma_x^2 \to \infty} \frac{\sigma_x^2}{\sigma_n^2 + \sigma_x^2} y = \hat{x}_{ML}$$

- To understand this concept better, let's look at the simulation of the estimated x under ML and MAP as the variance of the transmitted signal is changing while the variance of the noise is constant at $\sigma^2 = 10$.

Table 3.6.1

$\hat{\theta}_{ML}$	$\hat{\theta}_{MAP}$	σ_x^2	σ_n^2
13.23	6.62	10	10
9.80	7.84	20	10
11.16	10.73	50	10
37.99	37.62	100	10

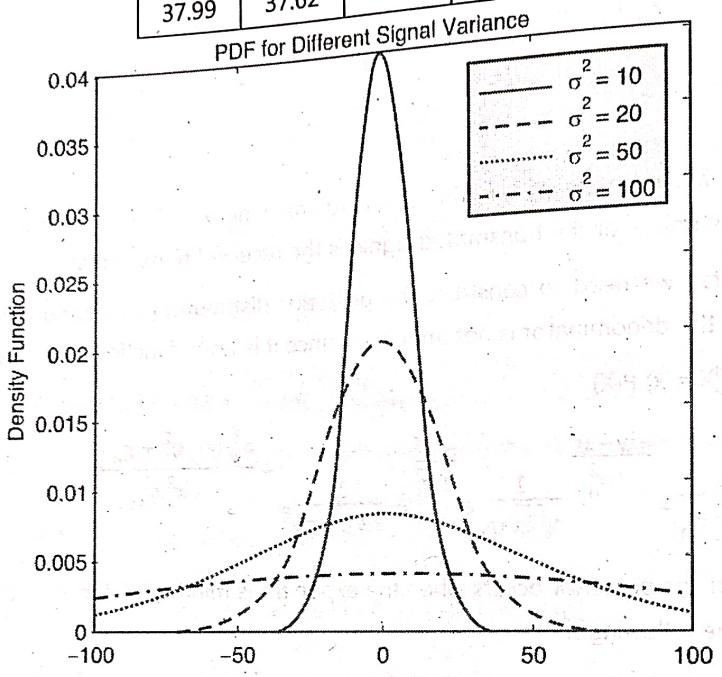


Fig. 3.6.1

- As the Table 3.6.1 and Fig. 3.6.1 shows the variance of the signal increases, the Gaussian distribution becomes more similar to a uniform distribution, and ML and MAP estimates become closer to each other. For example, when the signal variance is 10 times noise variance (The red curve) the ML and MAP estimates are almost identical.

2. Bayes Estimator

- In Bayesian estimation, the parameter θ is modeled as a random variable (Recall Bayesian vs Frequentists section in this article) with a certain probability distribution. The MMSE, LAE (Least Absolute Error), and MAP are all special types of Bayes estimators.
- Define the cost or loss function C as a cost of choosing the estimator instead of the true parameter (Think of it as how much you lose if you use the estimator instead of the true parameter). In the Bayes estimation, we minimize the expected loss function given observed data x. This can be defined mathematically as follows :

$$E[C(\hat{\theta}, \theta) \mid x] = \int C(\hat{\theta}, \theta) P(\theta \mid x) d\theta$$

- $P(\theta|x)$ is the posterior distribution (Refer to the first section in this article) and can be calculated using the Bayes law.
- The procedure to solve the Bayes estimation problems is as follows :
  1. Use the suitable cost function C.

2. After defining the cost function and simplifying the integrand, differentiate the expression, and set it to zero to find the estimator.

- The cost function can take many different forms however the most well-known cost functions are the quadratic and the absolute functions. For the rest of this section, we will derive the Bayes estimator for the quadratic, absolute, and 0–1 cost functions.

Quadratic Cost Function :

- If the cost function is quadratic, C is replaced by a quadratic function and follow the procedure outlined above.

$$J = E[C(\hat{\theta}, \theta \mid x)] \int (\theta - \hat{\theta})^2 P(\theta \mid x)\, d\theta$$

$$= \int (\theta^2 + \hat{\theta}^2 - 2\theta\hat{\theta}) P(\theta \mid x)\, d\theta$$

$$\frac{\partial J}{\partial \hat{\theta}} = 2\int \hat{\theta} P(\theta \mid x)\, d\theta - 2\int \theta P(\theta \mid x)\, d\theta = 0$$

$$\hat{\theta} \int P(\theta \mid x) d\theta = \int \theta P(\theta \mid x)$$

$$\hat{\theta} = E[\theta \mid x]$$

- **Interpretation :** The estimator is the conditional expectation of the parameter given the data or the posterior mean which is the identical result as the MMSE. Therefore the MMSE is the Bayes estimator when the cost function is quadratic.

Absolute Cost Function :

- In this case, C is replaced by the absolute error function. The estimator is calculated similarly :

$$J = E[C(\hat{\theta}, \theta \mid x)] = \int_{-\infty}^{\infty} |\theta - \hat{\theta}| P(\theta \mid x)\, d\theta - \int_{-\infty}^{\hat{\theta}} (\theta - \hat{\theta}) P(\theta \mid x)\, d\theta + \int_{\hat{\theta}}^{\infty} (\theta - \hat{\theta}) P(\theta \mid x)\, d\theta$$

$$\frac{\partial J}{\partial \hat{\theta}} = \int_{-\infty}^{\hat{\theta}} P(\theta \mid x)\, d\theta - \int_{\hat{\theta}}^{\infty} P(\theta \mid x)\, d\theta = 0$$

- Therefore, for the posterior distribution, the integral from -∞ to θ is equal to the integral from ∞ to θ. However, we also know that the integral over the entire domain would be 1 (Maximum value of probability is 1). Therefore:

$$-\int_{-\infty}^{\hat{\theta}} P(\theta \mid x)\, d\theta - \int_{\hat{\theta}}^{\infty} P(\theta \mid x)\, d\theta = 0$$

$$\int_{-\infty}^{\hat{\theta}} P(\theta \mid x)\, d\theta - \int_{\hat{\theta}}^{\infty} P(\theta \mid x)\, d\theta = 1$$

$$\int_{-\infty}^{\hat{\theta}} P(\theta \mid x)\, d\theta - \int_{\hat{\theta}}^{\infty} P(\theta \mid x)\, d\theta = \frac{1}{2}$$

- **Interpretation** : The best estimator under the absolute error cost function is the median of the posterior distribution. This is what one-half represents in the above equation. The estimator under this cost function is known as LAE (Least Absolute Error) estimators.

Zero-One Cost Function :

- In this case, C is 1 in some interval and zero otherwise.

$$C(\hat{\theta}, \theta) = \begin{cases} 1 & |\theta| \geq \theta_0 \\ 0 & |\theta| \leq \theta_0 \end{cases}$$

$$J = E[C(\hat{\theta}, \theta \mid x)] = \int_{-\infty}^{\infty} P(\theta \mid x)\, d\theta = \int_{-\infty}^{\theta_0} P(\theta \mid x)\, d\theta + \int_{\theta_0}^{\infty} P(\theta \mid x)\, d\theta$$

$$= 1 - \int_{-\theta_0}^{\theta_0} P(\theta \mid x)\, d\theta$$

$$\hat{\theta} = \arg\max P(\theta \mid x)$$

- **Interpretation** : Minimizing J is equivalent to maximizing the posterior distribution. Therefore the estimator of 0–1 loss function is the mode of the posterior distribution (The value of θ that maximizes the posterior distribution). This is exactly what MAP estimation does. This means MAP estimator is a Bayes estimator when the cost function is 0 – 1.

3.6.2(B) Smoothing

- Smoothing is the process of flattening a probability distribution implied by a language model so that all reasonable word sequences can occur with some probability. This often involves broadening the distribution by redistributing weight from high probability regions to zero probability regions.
- Smoothing not only prevents zero probabilities, attempts to improves the accuracy of the model as a whole.
- What should we do when terms from our vocabulary-words that we are familiar with-appear in a test set in a situation where we have never seen it before-say, following a word that they have never occurred after in training? We'll need to take a little amount of probability mass away from certain more frequent occurrences and give it to the unseen events in order to prevent a language model from giving them 0 probability.
- Smoothing or discounting is the name of this alteration. Laplace (add-one) smoothing, add-k smoothing, stupid backoff, and Kneser-Ney smoothing are a few examples of smoothing techniques.

Need of smoothing :

- In a language model, we use parameter estimation (MLE) on training data. We can't actually evaluate our MLE models on unseen test data because both are likely to contain words/n-grams that these models assign zero probability to. Relative frequency estimation assigns all probability mass to events in the training corpus. But we need to reserve some probability mass to events that don't occur (unseen events) in the training data.

Example :

- **Training data :** The cow is an animal.
- **Test data :** The dog is an animal.
- If we use **unigram model** to train;

$$P(\text{the}) = \frac{\text{Count(the)}}{\text{(Total number of words in training set)}} = \frac{1}{5}.$$

Likewise, $P(cow) = P(is) = P(an) = P(animal) = \frac{1}{5}$

- To evaluate (test) the **unigram model**;

$$P(\text{the cow is an animal}) = P(the) \times P(cow) \times P(is) \times P(an) \times P(animal) = 0.00032$$

- While we use unigram model on the test data, it becomes zero because P(dog) = 0. The term 'dog' never occurred in the training data. Hence, we use smoothing.

1. Laplace smoothing :

- Adding one to each of the n-gram counts before normalising them into probabilities is the simplest method for smoothing. All counts that were previously 0 will now all be counts of 1, counts of 1, and so on. The Laplace smoothing procedure is used in this. Laplace smoothing introduces many of the notions we find in other smoothing methods, provides a helpful baseline, and is also a good smoothing approach for other tasks like text categorization. However, it does not perform well enough to be employed in recent n-gram models.
- Let's begin by applying Laplace smoothing on the probability associated with unigrams.
- Remember that the unigram probability's unsmoothed maximum likelihood estimate of the word w_i is its count c_i normalized by the total number of word tokens N :

$$P(w_i) = \frac{c_i}{N}$$

- Laplace smoothing only increases each count by one (hence its alternate name add one smoothing). We also need to change the denominator to account for the additional V observations because there are V terms in the lexicon and each one was given an increment. (What happens to our P values if we don't increase the denominator?)

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$$

- It is easier to explain how a smoothing method impacts the numerator by establishing an adjusted count c* rather than modifying both the numerator and denominator. By normalising by N, this adjusted count may be converted into a probability similar to an MLE count, making it simpler to compare directly with MLE counts. Since we are simply modifying the numerator, we must additionally multiply the result by a normalisation factor, N/N+V, in order to define this count:

$$c_i^* = (c_i + 1)\frac{N}{N + V}$$

- We can now turn c_i^* into a probability P_i^* by normalizing by N.
- To derive the probability mass that will be allocated to the zero counts, one way to think of smoothing is as discounting (reducing) certain non-zero counts. The ratio of the discounted counts to the original counts, or relative discount, or d_c, might be used to describe a smoothing process instead of the discounted counts, or c*:

$$d_c = \frac{c^*}{c}$$

- Now that we have the intuition for the unigram case, let's smooth our Berkeley Restaurant Project bigrams. Fig. 3.6.2 shows the add-one smoothed counts for the bigrams.

Bigram count

Original :

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Smoothed :

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Fig. 3.6.2 : Bigram count

- As shown in Fig. 3.6.2, the Berkeley Restaurant Project corpus of 9332 utterances contains eight words (out of V = 1446) that are add-one smoothed bigram counts. Counts that had previously 0 are in grey.
- The add-one smoothed probability for the bigrams is displayed in Fig. 3.6.2. Remember that the unigram count is used to normalise each row of counts in order to calculate the normal bigram probabilities:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}\, w_n)}{C(w_{n-1})}$$

- We must increase the unigram count by the whole number of word types in the vocabulary V in order to get add-one smoothed bigram counts :

$$P_{laplace}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}\, w_n) + 1}{\sum_w (C(w_{n-1}) + 1)} = \frac{C(w_{n-1}\, w_n) + 1}{C(w_{n-1}) + V}$$

- Therefore, V = 1446 must be added to each of the unigram counts listed in the preceding section. The smoothed bigram probabilities shown in Fig. 3.6.2 are the end outcome.

Bigram Probabilities :

Original :

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.0079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Smoothed :

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.0056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Fig. 3.6.3 Bigram Probability

- As shown in Fig. 3.6.3, the BeRP corpus of 9332 phrases has add-one smoothed bigram probabilities for eight of the words (out of V = 1446). Probabilities that were once 0 are shaded in grey
- Reconstructing the count matrix makes it easy to see how much a smoothing method has altered the initial counts. The following equation is used to calculate these corrected counts. The reconstructed counts are displayed in Fig. 3.6.3.

$$c^*(w_{n-1}\,w_n) = \frac{[C((w_{n-1}\,w_n) + 1) \times C(w_{n-1})]}{C(w_{n-1}) + V}$$

Reconstituted bigram count :

Original :

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Reconstituted :

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.54	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	630	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Fig. 3.6.4 : Reconstituted bigram count

- As shown in Fig. 3.6.4, add-one reconstituted counts for eight words (of V = 1446) in the BeRP corpus of 9332 sentences. Previously-zero counts are in gray.
- Be aware that the counts have changed significantly as a result of add-one smoothing. Changed from 609 to 238 C (wanted)! This is also seen in probability space: In the smoothed situation, P(to|want) falls from.66 in the unsmoothed case to.26. The count for each prefix word has been dramatically lowered, as shown by the discount d (the ratio between new and old counts); the discount for the bigram "want to" is.39, while the discount for "Chinese food" is.10, a factor of 10!
- Due to an excessive amount of probability mass being transferred to all zeros, counts and probabilities abruptly shift.

Example of Bigram Model :

Let us Bigram model example we need a corpus and the test data. Let us assume that the following is a small corpus;

Training corpus :

- <s> I am from Vellore </s>
- <s> I am a teacher </s>
- <s> students are good and are from various cities</s>
- <s> students from Vellore do engineering</s>

Test data :

- <s> students are from Vellore </s>
- Let us find the Bigram probability of the given test sentence. I explained the solution in two methods, just for the sake of understanding. the second method is the formal way of calculating the bigram probability of sequence of words.

Method -1 : As per the Bigram model, the test sentence can be expanded as follows to estimate the bigram probability;

P(<s> students are from Vellore </s>)

= P (students | <s>) × P (are | students) × P (from | are) × P(Vellore | from) × P(</s> | Vellore)

- To estimate bigram probabilities, we can use the following equation;

$$P(w_n \mid w_{n-1}) = \frac{\text{count}(w_{n-1}, w_n)}{\text{count}(w_{n-1})}$$

$$P(\text{students} \mid <s>) = \frac{\text{count}(<s> \text{ students})}{\text{count}(<s>)} = \frac{2}{4} = \frac{1}{2}$$

[**Hint :** count of sentence start (<s>) = 4, count of string <s> students = 1]

$$P(\text{are} \mid \text{students}) = \frac{\text{count (students are)}}{\text{count (students)}} = \frac{1}{2}$$

[**Hint :** count of word **students** = 2, count of string **students are** = 1]

$$P(\text{from} \mid \text{are}) = \frac{\text{count (are from)}}{\text{count (are)}} = \frac{1}{2}$$

[**Hint :** count of word **are** = 2, count of string **are** from = 1]

$$P(\text{Vellore} \mid \text{from}) = \frac{\text{count (from Vellore)}}{\text{count (from)}} = \frac{2}{3}$$

[**Hint :** count of word **from** = 3, count of string **from Vellore** = 2]

$$P(</s> \mid \text{Vellore}) = \frac{\text{count (Vellore } </s>)}{\text{count (Vellore)}} = \frac{1}{2}$$

[**Hint :** count of word **Vellore** = 2, count of string **Vellore </s>** = 1]

P(<s> students are from Vellore </s>) = P(students | <s>) × P(are | students) × P(from | are) × P(Vellore | from) × P(</s> | Vellore)

$$= \frac{1}{4} \times \frac{1}{2} \times \frac{1}{2} \times \frac{2}{3} \times \frac{1}{2} = 0.0208$$

Method 2 : Formal way of estimating the bigram probability of a word sequence :

- The bigram probabilities of the test sentence can be calculated by constructing Unigram and bigram probability count matrices and bigram probability matrix as follows;

Unigram count matrix

<s>	students	are	from	Vellore
4	2	2	3	2

Bigram count matrix :

w_{n-1}	w_n				
	students	**are**	**from**	**Vellore**	**</s>**
<s>	1	0	0	0	0
students	0	1	1	0	0
are	0	0	1	0	0
from	0	0	0	2	0
Vellore	0	0	0	0	1

Bigram probability matrix (normalized by unigram counts)

w_{n-1}	w_n				
	students	**are**	**from**	**Vellore**	**</s>**
<s>	1/4	0/4	0/4	0/4	0/4
students	0/2	1/2	1/2	0/2	0/2
are	0/2	0/2	1/2	0/2	0/2
from	0/3	0/3	0/3	2/3	0/3
Vellore	0/2	0/2	0/2	0/2	1/2

P(<s> students are from Vellore </s>) = P(students | <s>) × P(are | students) × P(from | are)

× P(Vellore | from) × P(</s> | Vellore)

$$= \frac{1}{4} \times \frac{1}{2} \times \frac{1}{2} \times \frac{2}{3} \times \frac{1}{2} = 0.0208$$

The probability of the test sentence as per the bigram model is 0.0208.

3.6.3 Evaluating Language Models

- The easiest approach to assess a language model's performance is to embed it in an application and measure how much the application improves.
- Extrinsic evaluation refers to such end-to-end evaluation.
- Extrinsic assessment is the only way to determine whether a certain enhancement in a component will actually benefit the job at hand. Thus, in the case of voice recognition, we may compare the performance of two language models by running the speech recognizer twice, once with each language model, and determining which yields the better accurate transcription.
- Unfortunately, it may be highly expensive to run large NLP systems from beginning to finish. A measure that can be used to swiftly assess prospective advancements in a language model would be preferable. A metric for intrinsic assessment assesses a model's value without regard to its use.
- A test set is required for an intrinsic evaluation of a language model. The training corpus, also known as the training set or training corpus, is where an n-gram model gets its probabilities, just like many other statistical models in our industry. The effectiveness of an n-gram model may then be evaluated based on how well it performs on a test set or test corpus of unobserved data.
- We therefore divide the data into training and test sets, train the parameters of both models on the training set, and then evaluate how well the two trained models match the test set if we are given a corpus of text and wish to compare two distinct n-gram models.
- But what does "fit the test set" actually mean? The answer is straightforward: whatever model gives the test set a higher probability-meaning it predicts the test set with greater accuracy-is a better model. The more closely a pair of probabilistic models fits the test data or more accurately anticipates its specifics, and hence gives the test data a higher probability, the better the model.
- It's crucial to keep the test words out of the training set since our evaluation measure is dependent on the likelihood of the test set. Imagine that we are attempting to determine the likelihood of a specific "test" statement. When our test phrase appears in the test set, we will wrongly give it an unnaturally high probability if it is a part of the training corpus. The test set circumstance training is what we refer to as. The probability-based measure of perplexity, which we describe below, suffers greatly from the bias introduced by training on the test set, which makes all of the probabilities appear to be too high.
- Sometimes we employ a specific test set so frequently that we unconsciously adjust to its properties. Then, we require a brand-new, completely unknown test set. In these circumstances, the original test set is referred to as the development test set, or dev set. How can we create training, development, and test sets from our data? A tiny test set may unintentionally be unrepresentative; therefore, we want it to be as big as it can be.
- At the same time, we want as much training data as we can get. The smallest test set that provides us with sufficient statistical power to detect a statistically significant difference between two viable models is the one we should choose, at the very least. In practise, we frequently just split our data into three categories: training (80%), development (10%), and testing (10%).
- If we have a big corpus that we want to split into training and testing, we may either extract test data from a continuous sequence of text within the corpus or we can extract smaller "stripes" of text from several randomly chosen regions of our corpus and merge them into a test set.

Perplexity :

- In reality, we evaluate language models using a different measure called perplexity rather than pure probability. The perplexity of a language model on a test set is the inverse probability of the test set, normalised by the number of words. This term is sometimes abbreviated as PP. A test set $W = w_1 w_2 \ldots w_N$, as follows :

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-1/N}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \quad \ldots(3.6.1)$$

- The chain rule can be applied to increase the likelihood of W :

$$PP(W) = \sqrt[n]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \ldots w_{i-1})}} \quad \ldots(3.6.2)$$

- Consequently, if we use a bigram language model to calculate the perplexity of W, we obtain :

$$PP(W) = \sqrt[n]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1})}} \quad \ldots(3.6.3)$$

- The higher the conditional probability of the word sequence, the lower the perplexity, as shown by the inverse in Equation (3.6.2). According to the language model, lowering perplexity is the same as increasing the probability of the test set. The whole word sequence in a test set is often what we utilise for word order in Equations (3.6.2) or (3.6.3). We must factor in the begin- and end-sentence markers in the probability calculation because this sequence will span numerous sentence boundaries.
- The total number of word tokens N must also account for the end-of-sentence marker (but not the beginning-of-sentence marker). Perplexity may also be thought of as a language's weighted average branching factor.
- A language's branching factor is the number of potential words that might come after any given word. Consider the problem of identifying the English digits (zero, one, two,..., nine), given that each of the 10 digits occurs with an equal probability P = 1/10 in both some training sets and some test sets. This mini-confusion language's is really 10. Imagine a test string of N-digits in length and suppose that each digit happened in the training set with an equal chance to illustrate what I mean. Equation (3.6.2) will reveal the complexity.

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-1/N}$$

$$= \left(\frac{1}{10}^{N}\right)^{-f} = \frac{1}{10}^{-1} = 10 \quad \ldots(3.6.4)$$

- However, let's say that zero happens a lot more frequently than other numbers. Let's assume that in the training set, 0 appeared 91 times, whereas the other numbers appeared only once each. The following test set is now visible: 0 0 0 0 0 3 0 0 0 0.
- Since the next number will be zero most of the time, which is extremely predictable and so has a high probability, we should anticipate that the perplexity of this test set would be reduced. Consequently, the confusion or weighted branching factor is lower even if the branching factor is still 10.
- As is well known, entropy, a concept from information theory, and confusion are closely connected concepts.
- Let's have a look at an illustration of how perplexity may be utilised to contrast several n-gram models. Using a 19,979-word vocabulary, we trained unigram, bigram, and trigram grammars on 38 million words (including start-of-sentence tokens) from the Wall Street Journal.

- After that, using Equation (3.6.3), we determined the perplexity of each of these models on a test set of 1.5 million words. According to each of these grammars, the perplexity of a 1.5-million-word WSJ test set is displayed in the Table 3.6.1.

Table 3.6.1

	Unigram	Bigram	Trigram
Perplexity	962	170	109

- As we can see above, the confusion decreases as the n-gram provides us with more information about the word sequence (since as Equation (3.6.2) showed, perplexity is related inversely to the likelihood of the test sequence according to the model).
- It should be noted that while computing perplexities, the n-gram model P must be created without any prior knowledge of the test set or its vocabulary. Any type of test set knowledge can artificially lower the perplexity. Only if two language models employ the same vocabulary can their perplexities be compared.
- A (intrinsic) increase in perplexity does not always translate to a (extrinsic) increase in a language processing task's effectiveness, such as voice recognition or machine translation.
- Nevertheless, it is frequently employed as a rapid check on an algorithm because ambiguity frequently corresponds with such advancements. But before a model's assessment is finished, the perplexity improvement of the model should always be verified by an end-to-end evaluation of a real job.

3.7 Word Embeddings/ Vector Semantics

- Word Embedding is a technique of word representation that allows words with similar meaning to be understood by machine learning algorithms. Technically speaking, it is a mapping of words into vectors of real numbers using the neural network, probabilistic model, or dimension reduction on word co-occurrence matrix.
- Word embedding can be learned using a variety of language models. For example, a ***'dog'*** will be represented by the vector ***[0.75, 0.22, 0.66, 0.97]***. If all the words in a dictionary are encoded in such a way, it becomes possible to compare the vectors of the words with each other, for example by measuring the *cosine distance or the euclidean distance* between the vectors.
- A good representation of words will then make it possible to find that the word ***'pet'*** is closer to the word ***'dog'*** than it is to the word ***'soccer'*** or ***'engine'***. Therefore, these representations allow us to hope that, in the vector space where the embedding is made, we will have the equation ***king-man+woman = queen*** or even the equation ***London-England+Italy = Rome.***
- Word embeddings are also very useful in mitigating the curse of dimensionality, a very recurring problem in artificial intelligence. Without word embedding, the unique identifiers representing the words generate scattered data, isolated points in a vast sparse representation.
- With word embedding, on the other hand, the space becomes much more limited in terms of dimensionality with a widely richer amount of semantic information.
- With such numerical features, it is easier for a computer to perform different mathematical operations like matrix factorization, dot product, etc. which are mandatory to use shallow and deep learning techniques. There are many techniques available at our disposal to achieve this transformation.

3.7.1 Bag-of-words

- Bag-of-Words (*a.k.a.* BOW) is a popular basic approach to generate document representation. A text is represented as a bag containing plenty of words. The grammar and word order are neglected while the frequency is kept the same. A feature generated by bag-of-words is a vector where ***n*** is the number of words in the input documents vocabulary.
- For instance, there are two documents :
  - **doc_1 :** "I love Italian food and Tunisian food."
  - **doc_2 :** "I love the food here."
- The vocabulary of these two documents is :

  ["I", "love", "Italian", "food", "and", "Tunisian", "here",]
- This vocabulary will produce feature vectors of length **8** (*i.e.* Vocabulary Cardinality). Given such vocabulary, the bag-of-word feature representation of these two documents are:
  - **BOW(doc_1) :** [1,1,1,2,1,1,0,0]
  - **BOW(doc_2) :** [1,1,0,1,0,0,1,1]
- Using this technique we create the bag-of-words representation of each document in our dataset. If our dataset contains a big number of unique words some of that are not used very frequently, which is usually the case. So, among those, we chose N most frequent words and create a feature vector of dimension N x 1. These feature vectors are then used for any machine learning task.

3.7.2 TFIDF

- The tf-idf is a product of two terms: term frequency (tf) and inverse document frequency (idf). The TF defines the frequency of a given term in a given document. In an NLP application, since the frequency of a term might be too high, we down weight the frequency term by applying log10 scale. So, TF is defined as :

$$tf_{(t,d)} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{for count}(t,d) > 0 \\ 0 & \text{Else} \end{cases}$$

- Inverse Document Frequency (idf) assigns higher weights to words that occur only in few documents. Such words are quite useful for discriminating those documents from the rest of the collection. The idf is defined as N/dft, where N is number of documents and dft is document frequency - the number of documents in which the term (t) occurs. Here too, we apply the log10 scale to down weight the IDF value:

$$id\ f_t = \log_{10}(N / d\ f_t)$$

- We can now compute the tf-idf weight vector :

$$w_{t,d} = t\ f_{t,d} \times id\ f_t$$

- As you can see here, the tf-idf appropriately measures the importance of a word and helps us identify whether a given word is discriminative or not.
- As stated at the beginning, the biggest disadvantage with co-occurrence model is that it results in a long, sparse matrix.

- NLP applications typically prefer a dense matrix with lower sparsity. In spite of its sparse nature, the tf-idf vector model is still useful in applications such as Information retrieval. Traditionally, NLP specialists use TF-IDF model for baseline metric before trying out more advanced topics.

3.7.3 word2vec

- One of the most efficient techniques to represent a word is Word2Vec. Word2vec is a computationally efficient predictive model for learning word embeddings from raw text. It plots the words in a multi-dimensional vector space, where similar words tend to be close to each other. The surrounding words of a word provide the context to that word.
- Let's start with a high-level insight about where we're going. Word2Vec uses a well-known trick in deep learning. In this trick, we train a simple neural network with a single hidden layer to perform a fake task, without a real need of the results of the task we trained it on.
- Instead, the main objective consists of learning a representation of the input data which are actually the 'word vectors' gathered from the learned weights of the hidden layer. Such an approach may remind us of the way we train auto-encoders to perform dimensionality reduction.
- Word2Vec can rely on either one of two model architectures in order to produce a distributed representation of input words: Continuous Bag-of-Words (CBoW) or Continuous Skip-Gram as shown in the Fig. 3.7.1. Vector representation extracts semantic relationships based on the co-occurrence of words in the dataset.
- The CBoW and skip-gram models are trained using a binary classification to discriminate between the real target word and the other words in the same context. The accuracy at which the model predicts the words depend on how many times the model sees these words within the same context throughout the dataset.
- The hidden representation is changed by more words and context co-occurrences during the training process, which allows the model to have more future successful predictions, leading to a better representation of word and context in the vector space. Skip gram is much slower than CBOW, but performs more accurately with infrequent words.

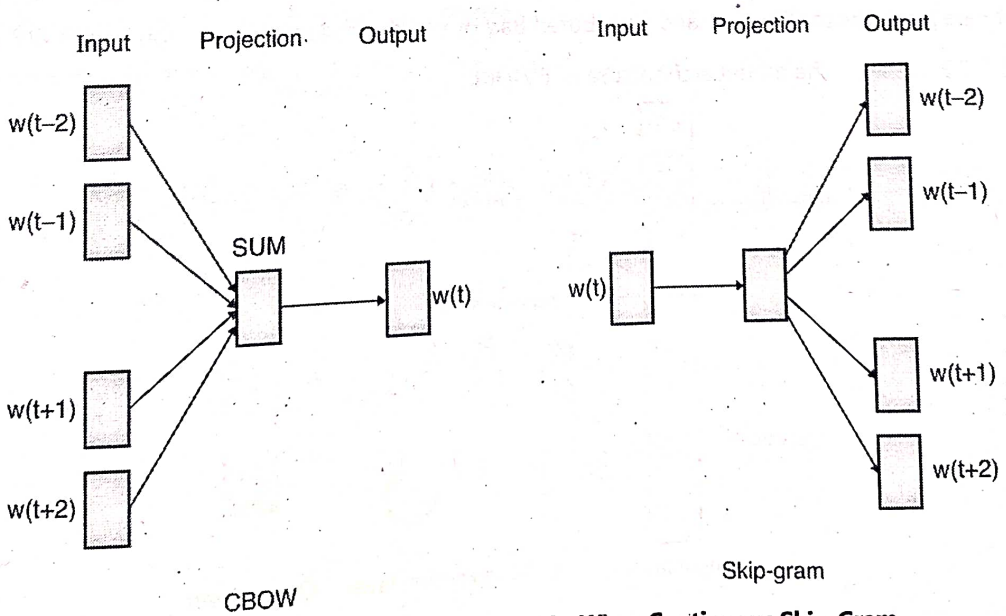


Fig. 3.7.1 Continuous Bag-of-Words (CboW) or Continuous Skip-Gram

Word2Vec Training Model Architecture

Let's go deeper into details to understand the difference between CBOW architecture and Continuous skip-gram.

- In the first option , CBOW; The model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). For example, if you gave the trained network the input words "Soccer", "NBA", "Game" and "Tennis", the output probabilities are going to be much higher for words like "Player" and "Tennis" then for unrelated words like "cheese" and "elections".
- Whereas, in the second option of using the continuous skip-gram architecture; the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words.
- The output probabilities are going to relate to how likely it is to find each vocabulary word near our input word. For example, if you gave the trained network the input word "Europe", the output probabilities are going to be much higher for words like "Belgium" and "Continent" than for unrelated words like "fruits" and "cats".
- In both models, when we say 'surrounding', there is actually a 'window size' parameter to the algorithm. The size of the context window limits how many words before and after a given word would be included as context words of the given word.
- For example, window size of 3 will include the 3 words to the left and the 3 words to the right for each observed word in the sentence as context. Increasing the window size increases the training time as more word-context pairs need to be trained. Also it may capture context words that are not relevant to the current word. Decreasing the context words can capture relations between words and stop words which is not preferred.

3.7.4 doc2vec

- Doc2Vec is another widely used technique that creates an embedding of a document irrespective to its length. While Word2Vec computes a feature vector for every word in the corpus, Doc2Vec computes a feature vector for every document in the corpus.
- Doc2vec model is based on Word2Vec, with only adding another vector (paragraph ID) to the input. The Doc2Vec model, by analogy with Word2Vec, can rely on one of two model architectures which are: Distributed Memory version of Paragraph Vector (PV-DM) and Distributed Bag of Words version of Paragraph Vector (PV-DBOW).
- In the Fig. 3.7.2, we show the model architecture of PV-DM :

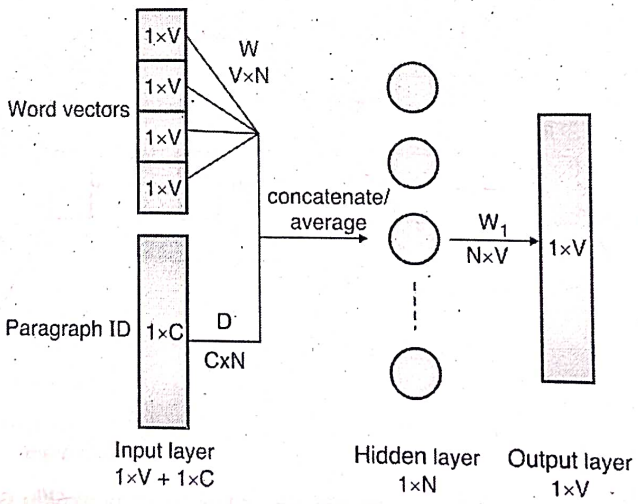


Fig. 3.7.2 PV-DM model architecture

Doc2Vec Model Architecture :

- The above diagram is based on the CBOW model, but instead of using just nearby words to predict the word, we also added another feature vector, which is document-unique. So when training the word vectors W, the document vector D is trained as well, and at the end of training, it holds a numeric representation of the document.
- The inputs consist of word vectors and document Id vectors. The word vector is a one-hot vector with a dimension 1 x V. The document Id vector has a dimension of 1 x C, where C is the number of total documents. The dimension of the weight matrix W of the hidden layer is N x V. The dimension of the weight matrix D of the hidden layer is C x N.

3.7.5 Contextualized representations (BERT)

- Bidirectional Encoder Representations from Transformers (BERT) is a popular deep learning model that is used for numerous different language understanding tasks. BERT shares the same architecture as a transformer encoder, and is extensively pre-trained on raw, unlabelled textual data using a self-supervised learning objective, before being fine-tuned to solve downstream tasks (e.g., question answering, sentence classification, named entity recognition, etc.).
- At the time of its proposal, BERT obtained a new state-of-the-art on eleven different language understanding tasks, prompting a nearly-instant rise to fame that has lasted ever since.
- BERT use three embeddings to compute the input representations. They are token embeddings, segment embeddings and position embeddings. "CLS" is the reserved token to represent the start of sequence while "SEP" separate segment (or sentence). Those inputs are
  - **Token embeddings :** general word embeddings. In short, it uses vector to represent token (or word). You can check out this story for detail.
  - **Segment embeddings :** sentence embeddings in another word. If input includes 2 sentence, corresponding sentence embeddings will be assigned to particular words. If input only include one sentence, one and only one sentence embeddings will be used. Segment embeddings is learnt before computing BERT. For sentence embeddings, you can check out this story for detail.
  - **Position embeddings :** Refer to the token sequence of input. Even if there are 2 sentences, position will be accumulated.

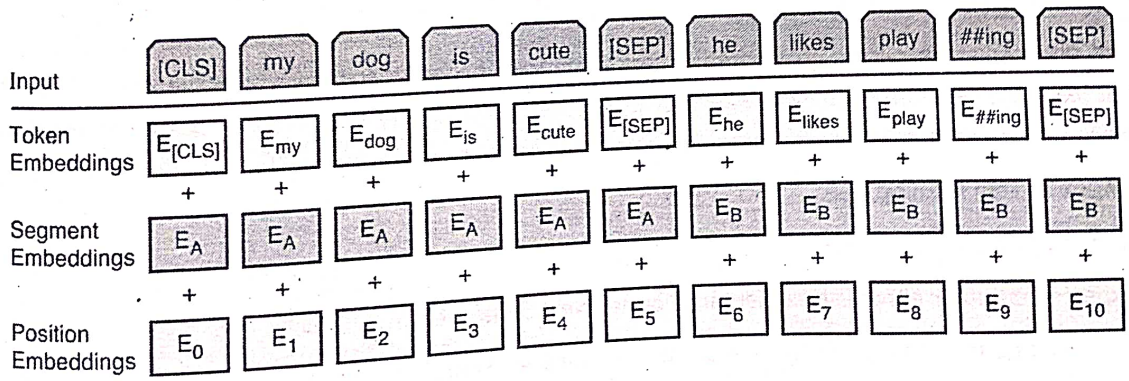


Fig. 3.7.3

BERT Input Representation (Devlin et al., 2018)

Training Tasks :

- After talking about input representation, I will introduce how BERT is trained. It uses two way to achieve it. First training task is masked language model while the second task is predicting next sentence.

Masked Language Model

- First pre-training tasks is leveraging masked language model (Masked LM). Rather than traditional directional model, BERT use bidirectional as a pre-training objective.
- If using traditional approach to train a bidirectional model, each word will able to see "itself" indirectly.
- Therefore, BERT use Masked Language Model (MLM) approach. By masking some tokens randomly, using other token to predicted those masked token to learn the representations. Unlike other approaches, BERT predict masked token rather than entire input.
- So the experiment pick 15% of token randomly to be replaced. However, there are some downsides. First disadvantage is that MASK token (actual token will be replaced by this token) will never seen in fine-tuning stage and actual prediction. Therefore, Devlin et al, the selected token for masking will not always be masked but
  - **A :** 80% of time, it will be replaced by [MASK] token
  - **B :** 10% of time, it will be replaced by other actual token
  - **C :** 10% of time, it will be keep as original.
- For example, the original sentence is "I am learning NLP". Assuming "NLP" is a selected token for masking. Then 80% of time, it will show as "I am learning [MASK] (Scenario A).
- "I am learning Open CV" in 10% of time (Scenario B). Rest of 10% of time, it will show as original which is "I am learning NLP" (Scenario C).
- Although random replacement (Scenario B) occur and may harming the meaning of sentence. But it is only 1.5% (Only mask 15% of token out of entire data set and 10% of this 15%) indeed, authors believe that it will not harm the model.
- Another downside is that only 15% token is masked (predicted) per batch, a longer time will take for training.

Next Sentence Prediction

- Second pre-training task is going to predict next sentence. This approach overcome the issue of first task as it cannot learn the relationship between sentences. The objective is very simple. Only classifying whether second sentence is next sentence or not. For example,
  - **Input 1 :** I am learning NLP.
  - **Input 2 :** NLG is part of NLP.
- The expected output is either is Next Sentence or not Next Sentence.
- When generating training data for this tasks, 50% of "not Next Sentence" data will be randomly selected.

Model Training :

- 2 phases training is applied in BERT. Using generic data set to perform first training and fine tuning it by providing domain specific data set.

Pre-training phase

- In pre-training phase, sentences are retrieved from Books Corpus (800M words) (Zhu et al., 2015) and English Wikipedia (2500M words).
  - Masked LM : 512 tokens per sequence (2 concatenated sentences) will be used and there are 256 sequences per batch. Approximate 40 epochs is set to train a model. The configure is :
  - Adam with learning rate of $1e^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$
  - L_2 weight decay of 0.01
  - 0.1 dropout for all layers
  - Using relu for activation
- As described before, two sentences are selected for "next sentence prediction" pre-training task. 50% of time that another sentence is pickup randomly and marked as "notNextSentence" wile 50% of time that another sentence is actual next sentence.
- This step done by Google research team and we can leverage this pre-trained model to further fine-tuning model based on own data.

Fine-tuning phase

- Only some model hyperpara meters are changed such as batch size, learning rate and number of training epochs, most mode hyperpara meters are kept as same in pre-training phase. During the experiments, the following range of value work well across tasks:
  - Batch Size : 16, 32
  - Learning Rate : $5e^{-5}$, $3e^{-5}$, $2e^{-5}$
  - Number of epochs : 3, 4
- Fine-tuning procedure is different and it depends on downstream tasks.

Classification

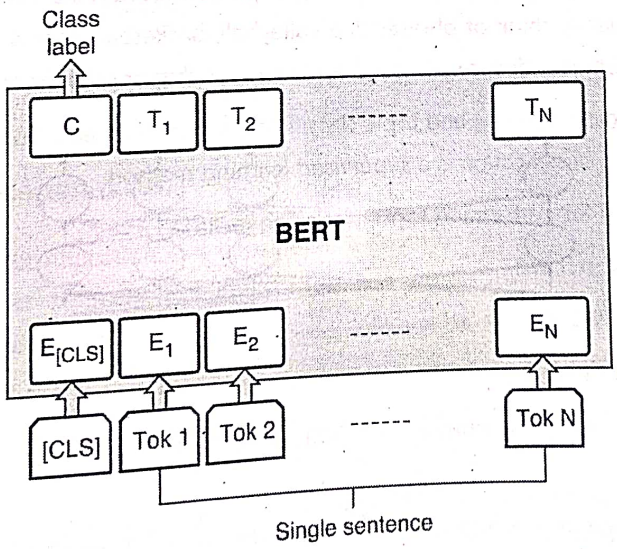


Fig. 3.7.4 : Single Sentence Classification Task (Devlin et al., 2018)

- For [CLS] token, it will be feed as the final hidden state. Label (C)probabilities are computed with a softmax. After that it is fine-tuned to maximize the log-probability of the correct label.

Named Entity Recognition

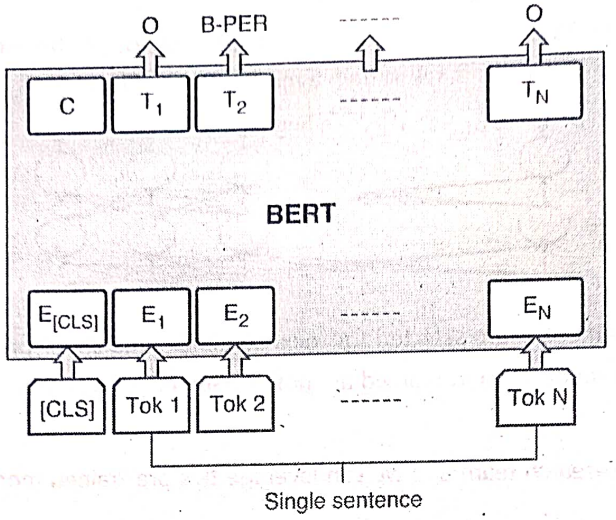


Fig. 3.7.5 : NER Task

- Final hidden representation of token will be feed into the classification layer. Surrounding words will be be considered on the prediction. In other words, the classification only focus on the token itself and no Conditional Random Field (CRF).

3.8 Topic Modelling

- Under unsupervised machine learning, topic modelling involves processing documents to identify relevant topics. Due to its ability to discover semantic relationships between words in document clusters, it is an important idea in the conventional Natural Processing Approach. It also has a wide range of other uses in NLP.
- Topics can be compared to keywords that can be used to define a document. For instance, when we think of the topic of sports, we immediately think of phrases like volleyball, basketball, tennis, and cricket. A topic model is a model that can identify topics in a document based on the words that appear there.
- The distinction between topic modelling and topic classification must be made. Topic modelling is an unsupervised learning process, while topic classification is a supervised learning method.
- Several well-known methods for performing topic modelling include :
  - Latent Dirichlet Allocation (LDA)
  - Latent Semantic Analysis (LSA)
  - Non-Negative Matrix Factorization (NMF)
- The above approaches represent a document as a bag-of-words and assume that each document is a mixture of latent topics.
- They all start with the conversion of a textual corpus into a Document -Term Matrix (DTM), a table where each row is a document, and each column is a distinct word :

Corpus	
Doucment 1	I like cats
Document 2	Cats are the best
Docuemt 3	Also dogs are nice

Document – Term Matrix									
	I	Like	Cats	Are	The	Best	Also	Dogs	Nice
Document 1	1	1	1	0	0	0	0	0	0
Document 2	0	0	1	1	1	1	0	0	0
Document 3	0	0	0	1	0	0	1	1	1

Document-Term Matrix from a sample set of documents.

Each cell <i, j> contains a count, i.e. how many times the word j appears in document i. A common alternative to the word count is the TF-IDF score. It considers both term frequency (TF) and inverse document frequency (IDF) to penalize the weight of terms that appear very often in the corpus, and increase the weight of more rare terms:

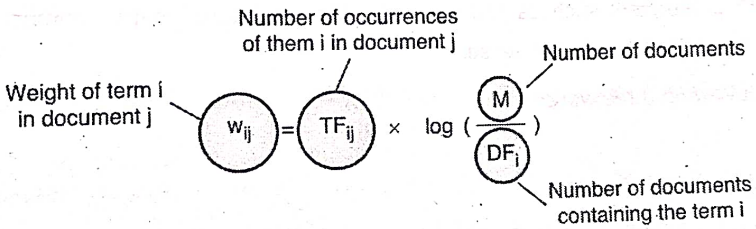


Fig. 3.8.1

TF-IDF :

The basic principle behind the search of latent topics is the decomposition of the DTM into a document-topic and a topic-term matrix.

3.8.1 Latent Dirichlet Allocation (LDA)

- LDA stands for Latent Dirichlet Allocation. It is considered a Bayesian version of pLSA. In particular, it uses priors from Dirichlet distributions for both the document-topic and word-topic distributions, lending itself to better generalization. It is a particularly popular method for fitting a topic model.
- The main assumption that LDA makes is that each document is generated by a statistical generative process i.e, each document is a mixture of topics, and each topic is a mixture of words. This algorithm exactly finding the weight of connections between documents and topics and between topics and words.
- Since this technique treats each document as a mixture of topics, and each topic as a mixture of words so this allows documents to "overlap" each other in terms of content, rather than being divided into discrete groups, in a way that mirrors the typical use of natural language.

- Since this technique treats each document as a mixture of topics, and each topic as a mixture of words so this allows documents to "overlap" each other in terms of content, rather than being divided into discrete groups, in a way that mirrors the typical use of natural language.

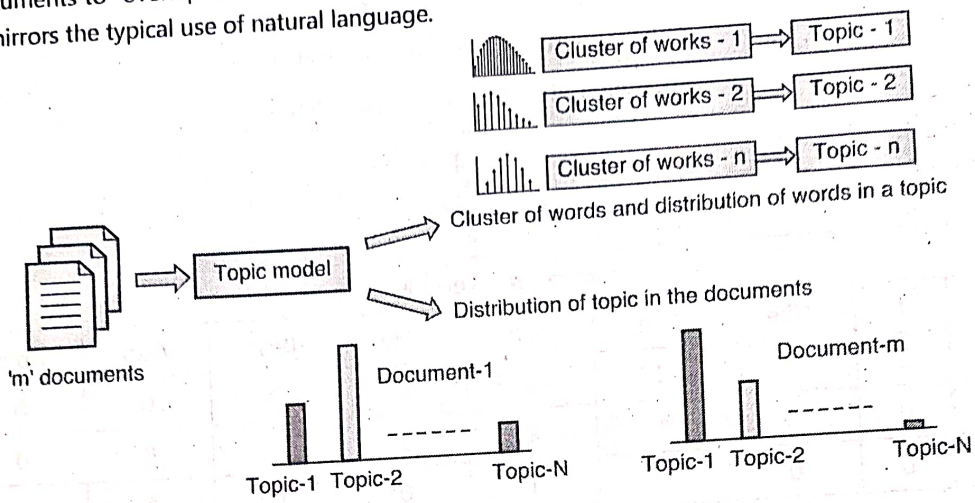


Fig. 3.8.2 LDA Model

Let's see what are the input and output to the LDA Algorithm?

- **Input :** Document-term matrix, number of topics, and number of iterations.
- **Output :** The top words in each topic. It is your job as a human to interpret this and see if the results make sense. If not, try altering the parameters such as terms in the document-term matrix, number of topics, number of iterations, etc. Stop when the topics make sense.
- LDA consists of the following three words :
  - Latent
  - Dirichlet
  - Allocation
- **'Latent'** represents that the model discovers the 'yet-to-be-found' or hidden topics from the documents.
- **'Dirichlet'** represents LDA's assumption that the distribution of topics in a document and the distribution of words in topics are both Dirichlet distributions.
- **'Allocation'** represents the distribution of topics in the document.

Importance of Dirichlet Distributions :

- It helps us to encode the intuition that documents are related to a few topics.

  Practically speaking, this leads to more accurate word disambiguation and topic assignment of papers. The documents would be evenly distributed over all the topics if we use the random distribution.
- This suggests that the likelihood of a paper being on one particular topic as opposed to all topics at once is equal. But we are aware that they are dispersed more thinly in reality. While there is still a chance that certain documents will pertain to more than one issue, it is clear that most of them are related to just one. The same holds true for topics and words. In order to model this kind of behaviour in a very natural way, Dirichlet distributions are helpful.

Let's examine the significance of the Dirichlet distribution using the following illustration :

- Consider the situation where we must compare topic mixes' probability distributions.
- Assume that the corpus we are concentrating on contains papers from 3 incredibly diverse subject areas. In these kinds of models, the distribution that gives a lot of weight to one particular topic and none at all to the other topics is what we want. For instance, if our corpus contains three topics, the following specific probability distributions might be helpful to see:
  - Mixture P : 90% topic A, 5% topic B, 5% topic C
  - Mixture Q : 5% topic A, 90% topic B, 5% topic C
  - Mixture R : 5% topic A, 5% topic B, 90% topic C
- Now, we would probably obtain a distribution that closely matches one of the mixtures from P, Q, or R if we randomly sampled a probability distribution from this Dirichlet distribution that is parameterized by big weights on a single issue. Additionally, there is a very small chance that we might sample a distribution that is 33% subject A, 33% topic B, and 33% topic C.
- This is essentially accomplished with the use of the Dirichlet distribution, a technique for sampling a certain kind of probability distribution.

3.8.1(A) Dirichlet Distributions

- A Dirichlet distribution Dir(α) is a way to model a Probability Mass Function, which gives probabilities for discrete random variables.
- Let's understand this with the help of an example of rolling a die, which we also discuss in the above section of the article.
  - It is a discrete random variable: The result is unpredictable, and the possible values can be 1, 2, 3, 4, 5, or 6.
  - If we have a fair dice, then a PMF would give the probabilities such as [0.16, 0.16, 0.16, 0.16, 0.16, 0.16]
  - If we have biased dice, then a PMF could return the probabilities: such as [0.25, 0.15, 0.15, 0.15, 0.15, 0.15], i.e, obtaining a one is higher than the other sides of the dice.
- In the example with documents, topics, and words, we'll have two PMFs :
  - Θ_{td} : the probability of topic *k* occurring in document *d*
  - Φ_{wt} : the probability of word *w* occurring in topic *k*
- The α in Dir(α) is known as the concentration parameter, and rules the trend of the distribution to be :
  - Uniform ($\alpha = 1$),
  - Concentrated ($\alpha > 1$),
  - Sparse ($\alpha < 1$)
- By using a concentration parameter $\alpha < 1$ the above probabilities will be closer to the real world. In other words, they follow Dirichlet distributions. Therefore,

  $\Theta_{td} \sim Dir(\alpha)$ and $\Phi_{wt} \sim Dir(\beta)$,

Probabilistic Approach of LDA

- LDA says is that each word in each document comes from a topic and the topic is selected from a per-document distribution over topics. So we have two matrices- one related to topic and document and the other related to word and topic. And, the probability of a word given document i.e. P(w|d) is equal to:

$$\sum_{t \in T} p(w \mid t, d)\, p(t \mid d)$$

where

T represents the total number of topics.

W represents the total number of words in our dictionary for all the documents.

- If we assume conditional independence, we can say that

$$P(w|t,d) = P(w|t)$$

- And hence the expression of P(w|d) reduces to :

$$\sum_{t=1}^{T} p(w \mid t)\, p(t \mid d)$$

i.e, it is the dot product of Θtd and Φ_{wt} for each topic t.

Now, let' see the step by step procedure of the probabilistic approach for LDA is shown below :

Step 1 : Go through each of the documents in a corpus and randomly assign each word in the document to one of K topics (K is chosen beforehand or given by the user).

Step 2 : With the help of random assignment, we got the topic representations for all the documents and word distributions of all the topics, but these are not very good ones.

Step 3 : So, to improve upon them

For each document d, we go through each word w and compute the following :

- **p(topic t | document d) :** represents the proportion of words present in document d that are assigned to topic t of the corpus.
- **p(word w | topic t) :** represents the proportion of assignments to topic t, over all documents d, that comes from word w.

Step 4 : Reassign word w a new topic t', where we choose topic t' with probability p(topic t' | document d)* p(word w | topic t')

This generative model predicts the probability that topic t' generate word w.

Step 5 : Repeating step-4 a large number of times, up to we reach a steady-state and at that state the topic assignments are pretty good. And finally, we use these assignments to determine the topic mixtures of each document.

Step 6 : After completing a certain number of iterations, we achieved a steady state where the document topic and topic term distributions are fairly good. And this becomes the convergence point of LDA.

- In the above process, if our guess of the weights is wrong, then the actual data that we observe will be very unlikely under our assumed weights and data generating process. Therefore, to resolve that issue we are trying to maximize the likelihood of our data given these two matrices.
- To identify the correct weights, we will use an algorithm known as Gibbs sampling.

Gibbs Sampling Algorithm

- It is an algorithm that can be used for successively sampling conditional distributions of variables, whose distribution over states converges to the true distribution in the long run.
- Here, In this blog post, we will not going to discuss the maths behind Gibbs Sampling but will try to give an intuition on how Gibbs Sampling work to identify topics in the documents.
- As we discussed earlier, we will assume that we know Θ and Φ matrices. Now we will slowly change these matrices and get to an answer that maximizes the likelihood of the data that we have. This process is done on a word-by-word basis by changing the topic assignment of one word.
- **Assumption :** We will assume that we don't know the topic assignment of the given word but we know the assignment of all other words in the text and we will try to find the topics that will be assigned to this word.
- In mathematical terms, we are trying to find the conditional probability distribution of a single word's topic assignment conditioned on the rest of the topic assignments.
- Ignoring all the mathematical calculations, we will get a conditional probability equation that looks like this for a single word *w* in document *d* that belongs to topic *k*:

$$p\left(z_{d,n} = k \mid \vec{z}_{-d,n}, \vec{w}, \alpha, \lambda\right) = \frac{n_{d,k} + \alpha_K}{\sum_{i}^{K} n_{d,i} + \alpha_i} \frac{v_{k,w\,d,n} + \lambda_{w_{d,n}}}{\sum_{i} V_{k,i} + \lambda_i}$$

Where : $\mathbf{n_{(d,k)}}$: represents how many time a document d use topic k

$\mathbf{v_{(k,w)}}$: represents how many times a topic k uses the given word

$\boldsymbol{\alpha_k}$: Dirichlet parameter for the document to topic distribution

$\boldsymbol{\lambda_w}$: Dirichlet parameter for the topic to word distribution

- If you observe carefully in the above expression, we divide that complete expression into two parts. Now, let's see the importance of each part.
- The first part tells us how much each topic is present in a document and the second part tells how much each topic likes a word.
- Here we have to note that for each word, we will get a vector of probabilities that will explain how likely this word belongs to each of the topics. In the above equation, we also observe that the Dirichlet parameters can also act as smoothing parameters when the terms n(d, k) or v(k, w) goes to zero which means that there will still be some possibility that the word will choose a topic going forward.
- After including Gibbs sampling in our probabilistic approach of LDA, the pictorial representation of all the steps is as follows :

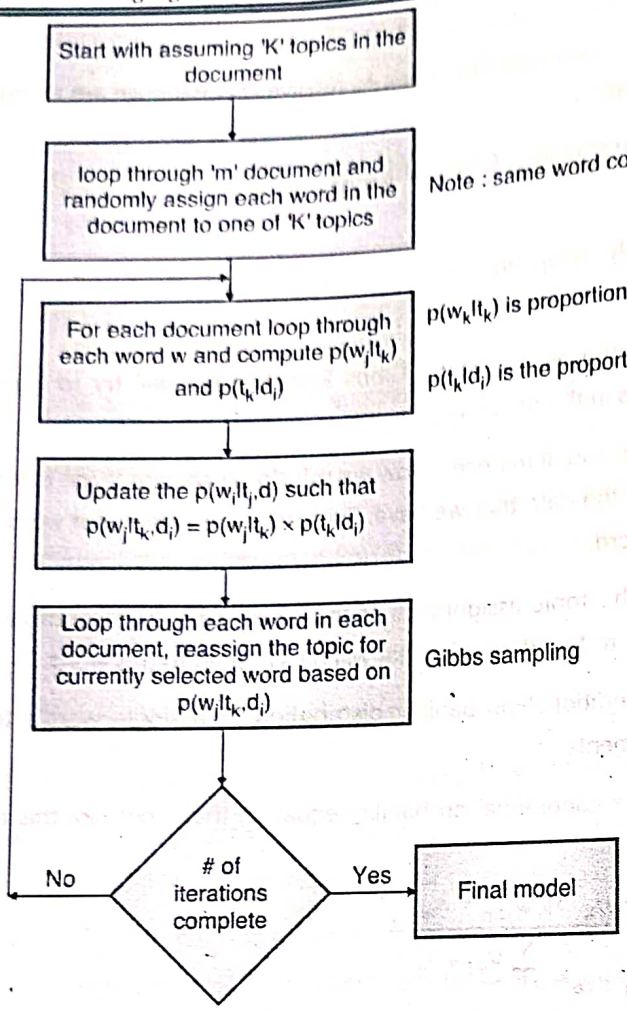


Fig. 3.8.3 : LDA Approach after including Gibbs Sampling

- Finally, as a concluding step or just outline the intuition behind the training process with Gibbs Sampling:
  - Randomly assign a topic to each word in the initial documents of the corpus.
  - Reassign the topic of each word such that each document contains the minimum possible topics.
  - Also, reassign the word of each topic such that each word is assigned the minimum possible topics.

where α and β rule each distribution and both have values < 1.

3.8.2 Latent Semantic Analysis

- Latent Semantic Analysis is another unsupervised learning approach for extracting relationships between words in a large number of documents. This assists us in selecting the appropriate documents.
- It merely serves as a dimensionality reduction tool for the massive corpus of text data. These extraneous data adds noise to the process of extracting the proper insights from the data.
- Latent Semantic Analysis is one of the natural language processing techniques for analysis of semantics, which in broad level means that we are trying to dig out some meaning out of a corpus of text with the help of statistical and was introduced by Jerome Bellegarde in 2005.

- LSA is basically a technique where we identify the patterns from the text document or in simple words we tend to find out relevant and important information from the text document. If we talk about whether it is supervised or unsupervised way, it is clearly an unsupervised approach.
- It is a very helpful technique in the reduction of dimensions of the matrix or topic modeling and is also known as **Latent Semantic Indexing(LSI).**
- The main concept and work of LSA are to group together all the words that have a similar meaning.

So how does it work? Let's see

Significance of Term Frequency/ Inverse Document Frequency in LSA :

1. Term Frequency is defined as a number of times instance or keyword appears in a single document divided by the total number of words in that document.

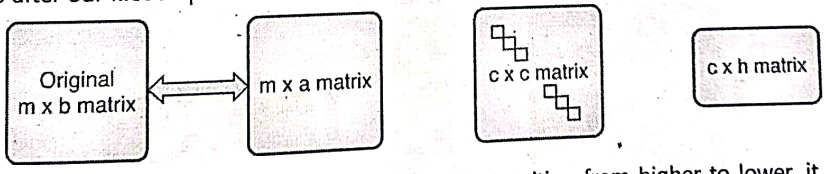
$$TF(t, d) = \frac{\text{Number of times term "t" appears in a document}}{\text{total Number of terms in a document "d"}}$$

As we know the length of the document is different in each case, so term frequency varies with the occurrence of term respectively.

2. Inverse Document Frequency (IDF), signifies how important the term is to be in the collection of documents. IDF calculates the weight of rare term of the text in a collection of documents. The formula of IDF is given by

$$IDF = \log \in \left(\frac{\text{Total number of document}}{\text{Number of documents that contains term "t"}}\right)$$

- The main idea of Tf/IDF in Latent Semantic Analysis is to provide each word count and the frequency of rare words in order to provide them weights on the basis of their rarity, TF/IDF is more preferable than conventional counting of occurrence of the word as it only counts the frequency without classification.
- After we have done the classification part using TF/IDF we tend to move to our next step that is the reduction of matrix dimension as normally with so many features the input have higher dimensions, a higher dimension input is hard to understand and interpret, so to lower the dimension with maximum information gain we have many techniques which include Singular Value Decomposition(SVD) and Principal Component Analysis. Let's see what SVD would do after our first step:-



- Singular value decomposition is a method for matrix decomposition from higher to lower, it usually divides the matrix into three matrices. Let us take an input matrix m x b of higher dimension as 'A', to calculate the SVD we will use the formula given below

$$A\ (m \times b) = U(m \times m) \cdot \sigma VT$$

- Here, σ is a diagonal matrix of size m x n and VT is a transpose of n x n orthogonal matrix. SVD may perform several other tasks but remains efficient primarily for dimension reduction, it is widely used and accepted by machine learning developers.

- Whenever SVD is performed, results are always classy, it can dramatically reduce more than 150k parameters or dimension to an understandable 50 to 70 parameters. With the completion of the above two tasks, it fulfills the motive of latent semantic analysis.
- There is much application of LSA to perform but it is mainly used in search engines as it is a very helpful technique there, for example, you searched 'sports' and the results also showed cricket and cricketers, this is due to LSA being applied on the search engines. Other possible applications of LSA are document clustering in text analysis, recommender systems and building user profiles.

3.8.3 Non-Negative Matrix Factorization (NMF)

Non-Negative Matrix Factorization is a statistical method that helps us to reduce the dimension of the input corpora or corpora. Internally, it uses the factor analysis method to give comparatively less weightage to the words that are having less coherence.

Some Important points about NMF :

1. It belongs to the family of linear algebra algorithms that are used to identify the latent or hidden structure present in the data.
2. It is represented as a non-negative matrix.
3. It can also be applied for topic modelling, where the input is the term-document matrix, typically TF-IDF normalized.
   - **Input :** Term-Document matrix, number of topics.
   - **Output :** Gives two non-negative matrices of the original n-words by k topics and those same k topics by the m original documents.
   - In simple words, we are using linear algebra for topic modelling.
4. NMF has become so popular because of its ability to automatically extract sparse and easily interpretable factors.

Below is the pictorial representation of the above technique :

$$A \approx W \times H$$

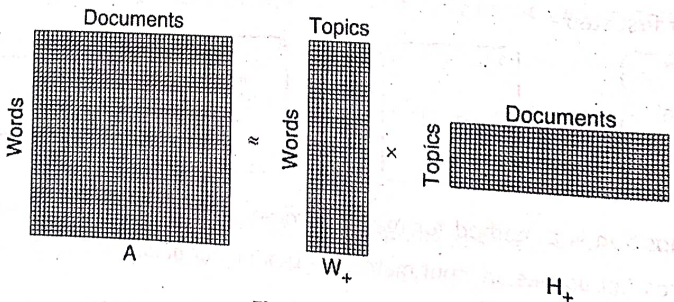


Fig. 3.8.4 : NMF

- As described in the image above, we have the term-document matrix (A) which we decompose it into two the following two matrices,
  - **First matrix :** It has every topic and what terms in it,
  - **Second matrix :** It has every document and what topics in it.

For Example :

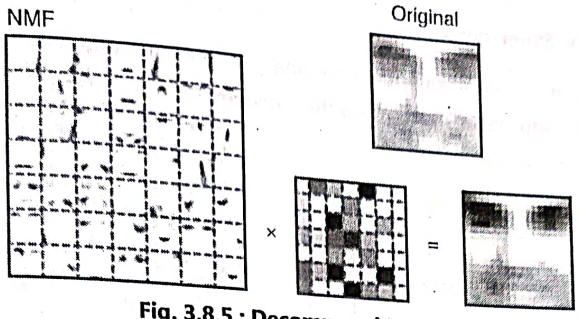
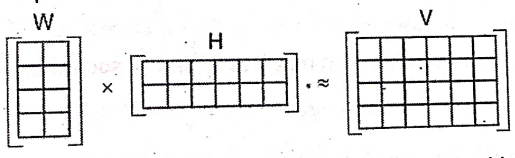


Fig. 3.8.5 : Decomposed Matrices

General case of NMF :

- Let's have an input matrix V of shape m x n. This method of topic modelling factorizes the matrix V into two matrices W and H, such that the shapes of the matrix W and H are m x k and k x n respectively.
- In this method, the interpretation of different matrices are as follows:
  - **V matrix :** It represents the term-document matrix,
  - **H matrix :** Each row of matrix H is a word embedding,
  - **W matrix :** Each column of the matrix W represents the weightage of each word gets in each sentence i.e, semantic relation of words with each sentence.
- But the main assumption that we have to keep in mind is that all the elements of the matrices W and H are positive given that all the entries of V are positive.



- Let's try to look at the practical application of NMF with an example described below :
  - Imagine we have a dataset consisting of reviews of superhero movies.
  - **Input matrix :** Here in this example, In the document term matrix we have individual documents along the rows of the matrix and each unique term along with the columns.
  - In case, the review consists of texts like Tony Stark, Ironman, Mark 42 among others. It may be grouped under the topic Ironman. In this method, each of the individual words in the document term matrix is taken into consideration.
  - While factorizing, each of the words is given a weightage based on the semantic relationship between the words. But the one with the highest weight is considered as the topic for a set of words. So this process is a weighted sum of different words present in the documents.

Maths behind NMF :

- As we discussed earlier, NMF is a kind of unsupervised machine learning technique. The main goal of unsupervised learning is to quantify the distance between the elements. To measure the distance, we have several methods but here in this blog post we will discuss the following two popular methods used by Machine Learning Practitioners :
  - Generalized Kullback–Leibler divergence
  - Frobenius norm

- Let's discuss each of them one by one in a detailed manner :

Generalized Kullback–Leibler Divergence

- It is a statistical measure that is used to quantify how one distribution is different from another. As the value of the Kullback–Leibler divergence approaches zero, then the closeness of the corresponding words increases, or in other words, the value of divergence is less.
- The formula for calculating the divergence is given by :

$$kl\_div(x, y) = \begin{cases} x \log(x/y) & x > 0, y > 0 \\ y & x = 0, y \geq 0 \\ \infty & \text{otherwise} \end{cases}$$

- Below is the implementation of Frobenius Norm in Python using Numpy:

Frobenius Norm

- It is another method of performing NMF. It is defined by the square root of the sum of absolute squares of its elements. It is also known as the euclidean norm. The formula for calculating the Frobenius Norm is given by :

$$\| A \|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{m} |a_{ij}|^2}$$

- It is considered a popular way of measuring how good the approximation actually is. Or if you want to find the optimal approximation to the Frobenius norm, you can compute it with the help of truncated Singular Value Decomposition (SVD).

Objective Function in NMF :

- Given the original matrix A, we have to obtain two matrices W and H, such that

$$A = WH$$

- NMF has an inherent clustering property, such that W and H described the following information about the matrix A :
  - **A (Document-word matrix) :** Input that contains which words appear in which documents.
  - **W (Basis vectors) :** The topics (clusters) discovered from the documents.
  - **H (Coefficient matrix) :** The membership weights for the topics in each document.
- Based on our prior knowledge of Machine and Deep learning, we can say that to improve the model and want to achieve high accuracy, we have an optimization process. There are two types of optimization algorithms present along with the scikit-learn package.
  - Coordinate Descent Solver
  - Multiplicative update Solver
- In this technique, we can calculate matrices W and H by optimizing over an objective function (like the EM algorithm), and updates both the matrices W and H iteratively until convergence.

$$\frac{1}{2} \| A - WH \|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} (A_{ij} - (WH)_{ij})^2$$

What exactly happens in this objective function ?

- In this objective function, we try to measure the error of reconstruction between the matrix A and the product of its factors W and H, on the basis of Euclidean distance.
- Now, by using the objective function, our update rules for W and H can be derived, and we get:

1. **For updation of elements of Matrix W :**

$$W_{ic} \leftarrow W_{ic} \frac{(AH)_{ic}}{(WHHH)_{ic}}$$

2. **For updation of elements of Matrix H :**

- Here we parallelly update the values and using the new matrices that we get after updation W and H, we again compute the reconstruction error and repeat this process until we converge.
- So, as a concluding step we can say that this technique will modify the initial values of W and H up to the product of these matrices approaches to A or until either the approximation error converges or the maximum iterations are reached.
- In our case, the high-dimensional vectors or initialized weights in the matrices are going to be TF-IDF weights but it can be really anything including word vectors or a simple raw count of the words.
- But there are some heuristics to initialize these matrices with the goal of rapid convergence or achieving a good solution. Now, in the next section let's discuss those heuristics.

Some heuristics to initialize the matrix W and H

- You can initialize W and H matrices randomly or use any method which we discussed in the last lines of the above section, but the following alternate heuristics are also used that are designed to return better initial estimates with the aim of converging more rapidly to a good solution.
  1. Use some clustering method, and make the cluster means of the top r clusters as the columns of W, and H as a scaling of the cluster indicator matrix (which elements belong to which cluster).
  2. Finding the best rank-r approximation of A using SVD and using this to initialize W and H.
  3. Picking r columns of A and just using those as the initial values for W.

Real-life Application of NMF :

- Image Processing uses the NMF. Let's look at more details about this.
- Say we have a gray-scale image of a face containing p number of pixels and squash the data into a single vector such that the i^{th} entry represents the value of the i^{th} pixel. Let the rows of $X \in R^{(p \times n)}$ represent the p pixels, and the n columns each represent one image.

Review Questions

Q.1 Explain probabilistic language model in detail ?

Q.2 Explain Bayes theorem with example ?

Q.3 Write short note on Hidden Markov Model ?

Q.4 Explain Generative models of language ?

Q.5 Explain Log-Liner Models ?

Q.6 Write short note on graph based model ?

Q.7 What is N-gram model ? Explain Unigram, Bigram and trigram mode l?

Q.8 Explain Parameter estimation techniques ?

Q.9 What is smoothing ? Explain in detail ?

Q.10 Explain Perplexity ?

Q.11 Write short note on Word Embeddings/ Vector Semantics ?

Q.12 Write short note on BERT ?

Q.13 Explain in detail topic modelling ?

Q.14 Explain. Latent Dirichlet Allocation (LDA) ?

Q.15 Explain Latent Semantic Analysis ?

Q.16 Explain Non-Negative Matrix Factorization (NMF) ?

□□□

4

Unit IV

Information Retrieval Using NLP

Syllabus

Information Retrieval: Introduction, Vector Space Model Named Entity Recognition: NER System Building Process, Evaluating NER System Entity Extraction, Relation Extraction, Reference Resolution, Coreference resolution, Cross Lingual Information Retrieval

4.1 Information Retrieval

4.1.1 Introduction

- Information retrieval is defined as the process of accessing and retrieving the most appropriate information from text based on a particular query given by the user, with the help of context-based indexing or metadata.
- **Google Search** is the most famous example of information retrieval. An information retrieval system searches a collection of natural language documents with the goal of retrieving exactly the set of documents that matches a user's question. They have their origin in library systems.
- These systems assist users in finding the information they require but it does not attempt to deduce or generate answers. It tells about the existence and location of documents that might consist of the required information that is given to the user. The documents that satisfy the user's requirement are called relevant documents. If we have a perfect IR system, then it will retrieve only relevant documents.

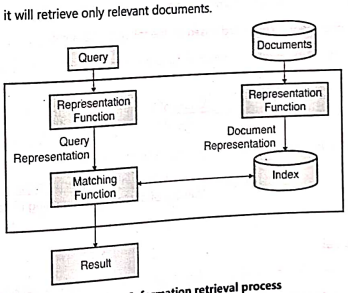


Fig. 4.1.1 : Information retrieval process

- From the above diagram, it is clear that a user who needs information will have to formulate a request in the form of a query in natural language. After that, the IR system will return output by retrieving the relevant output, in the form of documents, about the required information.

- The step by step procedure of these systems is as follows :
  - Indexing the collection of documents.
  - Transforming the query in the same way as the document content is represented.
  - Comparing the description of each document with that of the query.
  - Listing the results in order of relevancy.
- Retrieval Systems consist of mainly two processes :

1. Indexing
2. Matching

1. Indexing

- It is the process of selecting terms to represent a text.
- Indexing involves :
  - Tokenization of string
  - Removing frequent words
  - Stemming
- Two common Indexing Techniques :
  - Boolean Model
  - Vector space model

2. Matching :

- It is the process of finding a measure of similarity between two text representations.
- The relevance of a document is computed based on the following parameters :

1. **TF :** It stands for Term Frequency which is simply the number of times a given term appears in that document.

$$TF(i, j) = \frac{\text{(Count of ith term in } j^{th} \text{ document)}}{\text{(Total terms in } j^{th} \text{ document)}}$$

2. **IDF :** It stands for Inverse Document Frequency which is a measure of the general importance of the term.

$$IDF(i) = \frac{\text{(Total number of documents)}}{\text{(Number of documents containing } i^{th} \text{ term)}}$$

3. TF-IDF Score (i, j) = TF * IDF

4.1.2 Vector Space Model

- The vector space model is one of the most widely used models for ad-hoc retrieval, mainly because of its conceptual simplicity and the appeal of the underlying metaphor of using spatial proximity for semantic proximity.
- Documents and queries are represented in a high-dimensional space, in which each dimension of the space corresponds to a word in the document collection. The most relevant documents for a query are expected to be those represented by the vectors closest to the query, that is, documents that use similar words to the query.

- Rather than considering the magnitude of the vectors, closeness is often calculated by just looking at angles and choosing documents that enclose the smallest angle with the query vector.

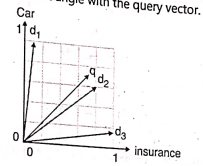


Fig. 4.1.2 : A vector space with two dimensions.

- The two dimensions correspond to the terms *car* and *insurance*. One query and three documents are represented in the space.
- In Fig. 4.1.2, we show a vector space with two dimensions, corresponding to the words **car** and **insurance.** The entities represented in the space are the query 4 represented by the vector (0.71, 0.71), and three documents $\mathbf{d_1}$, $\mathbf{d_2}$, and $\mathbf{d_3}$ with the following coordinates :

  (0.13, 0.99), (0.8, 0.6) and (0.99, 0.13).

- The coordinates or term **weights** are derived from occurrence counts as we will see below. For example, **insurance** may have only a passing reference in $\mathbf{d_1}$ while there are several occurrences of **car -** hence the low weight for **insurance** and the high weight for **car.** (In the context of information retrieval, the word **term** is used for both words and phrases. We say **term weights** rather than **word weights** because dimensions in the vector space model can correspond to phrases as well as words.)
- In Fig. 4.1.2, document $\mathbf{d_2}$ has the smallest angle with 9, so it will bathe top-ranked document in response to the query **car insurance.** This is because both concepts (car and insurance) are salient in $\mathbf{d_2}$ and therefore have high weights. The other two documents also mention both terms, but in each case one of them is not a centrally important term in the document.

Vector similarity :

- To do retrieval in the vector space model, documents are ranked according to similarity with the query as measured by the **cosine** measure or **normalized correlation coefficient.** The cosine as a measure of vector similarity and its definition here :

$$\cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \sqrt{\sum_{i=1}^{n} d_i^2}}$$

where $\vec{q}$ and dare n-dimensional vectors in a real-valued space, the space of all terms in the case of the vector space model. We compute how well the occurrence of term I (measured by q_i and d_i) correlates in query and document and then divide by the Euclidean length of the two vectors to scale for the magnitude of the individual q_i and d_i. cosine and Euclidean distance give rise to the same ranking for normalized vectors :

$$\left(|\vec{x} - \vec{y}|\right)^2 = \sum_{i=1}^{n} (x_i - y_i)^2 = \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} y_i^2$$

$$= 1 - 2\sum_{i=1}^{n} x_i y_i + 1$$

$$= 2\left(1 - \sum_{i=1}^{n} x_i y_i\right)$$

- So for a particular query $\vec{q}$ and any two documents $\vec{d}_1$ & and $\vec{d}_2$ and we have :

$$\cos(\vec{q}, \vec{d}_1) > \cos(\vec{q}, \vec{d}_2) \Leftrightarrow |\vec{q} - \vec{d}_1| < |\vec{q} - \vec{d}_2|$$

which implies that the rankings are the same.

- If the vectors are normalized, we can compute the cosine as a simple dot product. Normalization is generally seen as a good thing – otherwise longer vectors (corresponding to longer documents) would have an unfair advantage and get ranked higher than shorter ones.

Term weighting :

- We now turn to the question of how to weight words in the vector space model. One could just use the count of a word in a document as its term weight, but there are more effective methods of term weighting. The basic information used in term weighting is term *frequency, document frequency,* and sometimes collection frequency as defined in Table 4.1.1

Table 4.1.1 : Three quantities that are commonly used in term weighting in information retrieval

Quantity	Symbol	Definition
Term frequency	$tf_{i,j}$	Number of occurances of w_i in d_j
Document frequency	df_i	Number of documents in the collection that w_i occurs in
Collection frequency	cf_i	Total number of occurances of w_i in the collection

Table 4.1.2 : Term and document frequencies of two words in an example corpus.

Word	Collection	Frequency	Document	Frequency
Insurance		10440		3997
Try		10422		8760

- Note that $df_i <= Cf_i$ and that $\sum j\ tf_{ij} = cf_i$. It is also important to note that document frequency and collection frequency can only be used if there is a collection. This assumption is not always true, for example if collections are created dynamically by selecting several databases from a large set (as may be the case on one of the large on-line information services), and joining them into a temporary collection.
- The information that is captured by term frequency is how salient a word is within a given document. The higher the term frequency (the more often the word occurs) the more likely it is that the word is a good description of the content of the document.

- Term frequency is usually dampened by a function like $f(tf) = \sqrt{tf}$ or $f(tf) = 1 + \log(tf)$, $ft > 0$ or $f(tf) = 1 + \log(tf)$, $tf > 0$ because more occurrences of a word indicate higher importance, but not as much relative importance as the undampened count would suggest.
- For example, $\sqrt{3}$ or $1 + \log 3$ better reflect the importance of a word with three occurrences than the count 3 itself. The document is somewhat more important than a document with one occurrence, but not three times as important.
- The second quantity, document frequency, can be interpreted as an indicator of in formativeness. A semantically focused word will often occur several times in a document if it occurs at all. Semantically unfocussed words are spread out homogeneously over all documents. An example from a corpus of ***New York Times*** articles is the words ***insurance*** and try in Table 4.1.2
- The two words have about the same collection frequency, the total number of occurrences in the document collection. But **insurance** occurs in only half as many documents as **try**. This is because the word try can be used when talking about almost any topic since one can try to do something in any context. In contrast, insurance refers to a narrowly defined concept that is only relevant to a small set of topics.
- Another property of semantically focused words is that, if they come up once in a document, they often occur several times. ***Insurance*** occurs bout three times per document, averaged over documents it occurs in at least once. This is simply due to the fact that most articles about health insurance, car insurance or similar topics will refer multiple times to the concept of insurance.
- One way to combine a words term frequency tf_{ij} and document frequency df_i into a single weight is as follows :

$$\text{weight}(i,j) = \begin{cases} (1 + \log(tf_{i,j})) \log \frac{N}{df_i} & \text{if } tf_{i,j} \geq 1 \\ 0 & \text{if } tf_{i,j} = 0 \end{cases}$$

where N is the total number of documents. The first clause applies forwards occurring in the document, whereas for words that do not appear($tf_{ij} = 0$), we set weight(i, j) = 0.

Document frequency is also scaled logarithmically.

- The formula $\log \frac{N}{df_i} = \log N - \log df_i$ gives full weight to words that occur in 1 document ($\log N - \log d\ f_i = \log N - \log 1 = \log N$). A word that occurred in all documents would get zero weight ($\log N - \log d\ fi = \log N = 0$)
- This form of document frequency weighting is often called **inverse document frequency** or **idf** weighting. More generally, the weighting scheme in equation (weight (i, j)) is an example of a larger family of so-called **tf.df** weighting schemes. Each such scheme can be characterized by its term occurrence weighting, its document frequency weighting and its normalization.

4.2 Named Entity Recognition

Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on. Table 4.2.1 lists some of the more commonly used types of NEs. These should be self-explanatory, except for "FACILITY": human-made artifacts in the domains of architecture and civil engineering; and "GPE": geo-political entities such as city, state/province, and country.

Table 4.2.1 : Commonly used types of named entity

NE type	Examples
ORGANIZATION	*Georgia-Pacific Corp., WHO*
PERSON	*Eddy Bonte, President Obama*
LOCATION	*Murray River, Mount Everest*
DATE	*June, 2008-06-29*
TIME	*two fifty a m, 1:30 p.m.*
MONEY	*175 million Canadian Dollars, GBP 10.40*
PERCENT	*twenty pct, 18.75 %*
FACILITY	*Washington Monument, Stonehenge*
GPE	*South East Asia, Midlothian*

- The goal of a **Named Entity Recognition** (NER) system is to identify all textual mentions of the named entities. This can be broken down into two subtasks : identifying the boundaries of the NE, and identifying its type.
- While named entity recognition is frequently a prelude to identifying relations in Information Extraction, it can also contribute to other tasks.
- For example, in Question Answering (QA), we try to improve the precision of Information Retrieval by recovering not whole pages, but just those parts which contain an answer to the user's question. Most QA systems take the documents returned by standard Information Retrieval, and then attempt to isolate the minimal text snippet in the document containing the answer.
- Now suppose the question was *Who was the first President of the US ?*, and one of the documents that was retrieved contained the following passage :
- The Washington Monument is the most prominent structure in Washington, D.C. and one of the city's early attractions. It was built in honor of George Washington, who led the country to independence and then became its first President.
- Analysis of the question leads us to expect that an answer should be of the form *X was the first President of the US*, where *X* is not only a noun phrase, but also refers to a named entity of type PER. This should allow us to ignore the first sentence in the passage.
- Although it contains two occurrences of *Washington*, named entity recognition should tell us that neither of them has the correct type. How do we go about identifying named entities? One option would be to look up each word in an appropriate list of names.
- For example, in the case of locations, we could use a **gazetteer**, or geographical dictionary, such as the Alexandria Gazetteer or the

- Getty Gazetteer. However, doing this blindly runs into problems, as shown in Fig. 4.2.1.

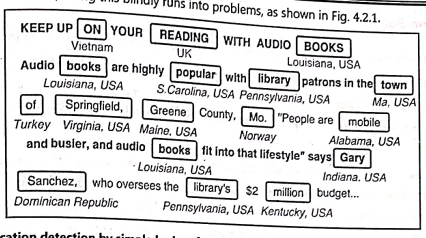


Fig. 4.2.1 : Location detection by simple lookup for a news story: Looking up every word in a gazetteer is error-prone; case distinctions may help, but these are not always present.

- Observe that the gazetteer has good coverage of locations in many countries, and incorrectly finds locations like Sanchez in the Dominican Republic and On in Vietnam. Of course we could omit such locations from the gazetteer, but then we won't be able to identify them when they do appear in a document.
- It gets even harder in the case of names for people or organizations. Any list of such names will probably have poor coverage. New organizations come into existence every day, so if we are trying to deal with contemporary newswire or blog entries, it is unlikely that we will be able to recognize many of the entities using gazetteer lookup.
- Another major source of difficulty is caused by the fact that many named entity terms are ambiguous. Thus May and North are likely to be parts of named entities for DATE and LOCATION, respectively, but could both be part of a PERSON; conversely Christian Dior looks like a PERSON but is more likely to be of type ORGANIZATION. A term like Yankee will be an ordinary modifier in some contexts, but will be marked as an entity of type ORGANIZATION in the phrase Yankee infielders.
- Further challenges are posed by multiword names like Stanford University, and by names that contain other names, such as Cecil H. Green Library and Escondido Village Conference Service Center. In named entity recognition, therefore, we need to be able to identify the beginning and end of multitoken sequences.
- Named entity recognition is a task that is well suited to the type of classifier-based approach that we saw for noun phrase chunking. In particular, we can build a tagger that labels each word in a sentence using the IOB format, where chunks are labeled by their appropriate type. Here is part of the CONLL 2002 Dutch training data :
  - Eddy N B-PER
  - Bonte N I-PER
  - is V O
  - woordvoerder N O
  - van Prep O
  - diezelfdePron O
  - Hogeschool N B-ORG
  - Punc O

4.2.1 NER System Building Process

- The first step in information extraction is to detect the entities in the text. A named entity is, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization.
- The term is commonly extended to include things that aren't entities per se, including dates, times, and other kinds of temporal expressions, and even numerical expressions like prices. Here's the sample text introduced earlier with the named entities marked :
  - Citing high fuel prices, [ORG United Airlines] said [TIME Friday] it has increased fares by [MONEY $6] per round trip on flights to some cities also served by lower-cost carriers. [ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said. [ORG United], a unit of [ORG UAL Corp.], said the increase took effect [TIME Thursday] and applies to most routes where it competes against discount carriers, such as [LOC Chicago] to [LOC Dallas] and [LOC Denver] to [LOC San Francisco].
- The text contains 13 mentions of named entities including 5 organizations, 4 locations, 2 times, 1 person, and 1 mention of money.
- In addition to their use in extracting events and the relationship between participants, named entities are useful for many other language processing tasks. In sentiment analysis we might want to know a consumer's sentiment toward a particular entity. Entities are a useful first stage in question answering, or for linking text to information in structured knowledge sources like Wikipedia.
- Table 4.2.2 shows typical generic named entity types. Many applications will also need to use specific entity types like proteins, genes, commercial products, or works of art.

Table 4.2.2 : A list of generic named entity types with the kinds of entities they refer to

Type	Tag	Sample categories	Example sentences
People	PER	People, characters	Turing is giant of computer science
Orgnization	ORG	Companies, sports teams	The IPCC warned abouth the cyclone
Location	LOC	Regions, mountaines, seas	The Mt. Saintas loopis in sunshine canyon
Geo-Political Entity	GPE	Countries, staes, provinces	Palo Alot is raising the fees for parking
Facility	FAC	Bridges, building, airports	Consider the Golden Gate Bridge.
Vehicles	VEH	Planes, trains, automobiles	It was classic Ford Falcon

- Named entity recognition means finding spans of text that constitute proper names and then classifying the type of the entity. Recognition is difficult partly because of the ambiguity of segmentation; we need to decide what an entity is and what isn't, and where the boundaries are.
- Another difficulty is caused by type ambiguity.
- The mention JFK can refer to a person, the airport in New York, or any number of schools, bridges, and streets around the United States. Some examples of this kind of cross-type confusion are given in Tables 4.2.3 and 4.2.4

Table 4.2.3 : Common categorical ambiguities associated with various proper names

Name	Possible Categories
Washington	Person, Location, Political Entity, Organization, vehicle
Downing St.	Location, Organization
IRA	Person, Organization, Monetary Instrument
Louis Vuitton	Person, Organization, Commercial Product

Table 4.2.4 : Examples of type ambiguities in the use of the name Washington

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.
The [VEH Washington] had proved to be a leaky ship, every passage I made....

4.2.1(A) NER as Sequence Labeling

- The standard algorithm for named entity recognition is as a word-by-word sequence labeling task, in which the assigned tags capture both the boundary and the type.
- A sequence classifier like an MEMM/CRF, a bi-LSTM, or a transformer is trained to label the tokens in a text with tags that indicate the presence of particular kinds of named entities. Consider the following simplified excerpt from our running example.

  [ORG American Airlines], a unit of [ORG AMRCorp.], immediately matched the move, spokesman [PER Tim Wagner] said.
- Table 4.2.5 shows the same excerpt represented with IOB tagging. In IOB tagging we introduce a tag for the beginning (B) and inside (I) of each entity type, and one for tokens outside (O) any entity.
- The number of tags is thus 2n + 1 tags, where n is the number of entity types. IOB tagging can represent exactly the same information as the bracketed notation.

Table 4.2.5 : Named entity tagging as a sequence model, showing IOB and IO encodings

Words	IOB Label	IO Label
American	B-ORG	I-ORG
Airlines	I-ORG	I-ORG
,	O	O
a	O	O
unit	O	O
of	O	O

Words	IOB Label	IO Label
AMR	B-ORG	I-ORG
Corp.	I-ORG	I-ORG
,	O	O
Immediately	O	O
Matched	O	O
the	O	O
move	O	O
,	O	O
spokesman	O	O
Tim	B-PER	I-PER
Wagner	I-PER	I-PER
said	O	O
.	O	O

- We've also shown IO tagging, which loses some information by eliminating the B tag. Without the B tag IO tagging is unable to distinguish between two entities of the same type that are right next to each other.
- Since this situation doesn't arise very often (usually there is at least some punctuation or other deliminator), IO tagging may be sufficient, and has the advantage of using only n+1 tags.
- In the following three sections we introduce the three standard families of algorithms for NER tagging: feature based (MEMM/CRF), neural (bi-LSTM), and rule-based.

4.2.1(B) A Feature-based Algorithm for NER

Identity of w_i, identity of neighboring words
embeddings for w_i, embeddings for neighboring words
part of speech of w_i, part of speech of neighboring words
base-pharse syntactic chunk label of w_i and neighboring words presence of w_i in a gazetter
w_i contains a particular perfix (for all perfixes of length ≤ 4)
w_i contains a particular suffix (form all suffixes of length ≤ 4)
w_i is all upper case
word shape of w_i, word shape of neighboring words
short word shape of w_i, short word shape of neighboring words presence of hyphen

Fig. 4.2.2 : Typical features for a feature-based NER system

- The first approach is to extract features and train an MEMM or CRF sequence model of the type we saw for part-of-speech tagging.
- Fig. 4.2.2 lists standard features used in such feature-based systems. We've seen many of these features before in the context of part-of-speech tagging, particularly for tagging unknown words. This is not surprising, as many unknown words are in fact named entities. Word shape features are thus particularly important in the context of NER.
- Recall that word shape features are used to represent the abstract letter pattern of the word by mapping lower-case letters to 'x', upper-case to 'X', numbers to 'd', and retaining punctuation. Thus for example I.M.F would map to X.X.X. and DC10-30 would map to XXdd-dd.
- A second class of shorter word shape features is also used. In these features consecutive character types are removed, so DC10-30 would be mapped to Xd-d but I.M.F would still map to X.X.X.
- This feature by itself accounts for a considerable part of the success of feature-based NER systems for English news text. Shape features are also particularly important in recognizing names of proteins and genes in biological texts.
- For example the named entity token *L' Occitane* would generate the following non-zero valued feature values :

  perfix (w_i) = L suffix (w_i) = tane
  perfix (w_i) = L' suffix (w_i) = ane
  perfix (w_i) = L'O suffix (w_i) = ne
  perfix (w_i) = L' Oc suffix (w_i) = e
  word shape (w_i) = X'Xxxxxxx short word shape (w_i) = X'Xx

- A gazetteer is a list of place names, often providing millions of entries for locations with detailed geographical and political information.
- A related resource is name-lists; the United States Census Bureau also provides extensive lists of first names and surnames derived from its decadal census in the U.S.2 Similar lists of corporations, commercial products, and all manner of things biological and mineral are also available from a variety of sources. Gazetteer and name features are typically implemented as a binary feature for each name list.
- Unfortunately, such lists can be difficult to create and maintain, and their usefulness varies considerably. While gazetteers can be quite effective, lists of persons and organizations are not always helpful.
- Feature effectiveness depends on the application, genre, media, and language. For example, shape features, critical for English newswire texts, are of little use with automatic speech recognition transcripts, or other non-edited or informally edited sources, or for languages like Chinese that don't use orthographic case.
- The features in Fig. 4.2.2 should therefore be thought of as only a starting point. Table 4.2.6 illustrates the result of adding part-of-speech tags, syntactic base-phrase chunk tags, and some shape information to our earlier example.
- Given such a training set, a sequence classifier like an MEMM can be trained to label new sentences. Fig. 4.2.3 illustrates the operation of such a sequence labeller at the point where the token Corp. is next to be labelled.
- If we assume a context window that includes the two preceding and following words, then the features available to the classifier are those shown in the boxed area.

Table 4.2.6 : Word-by-word feature encoding for NER

Word	POS	Chunk	Short-shape	Label
American	NNP	B-NP	Xx	B-ORT
Airlines	NNPS	I-NP	Xx	I-ORG
,	,	O	,	O
a	DT	B-NP	x	O
unit	NN	I-NP	x	O
of	IN	B-NP	x	O
AMR	NNP	B-NP	X	B-ORG
Corp.	NNP	I-NP	Xx,	I-ORG
,	,	O	,	O
Immediately	RB	B-ADVP	x	O
Matched	VBD	B-VP	x	O
the	DT	B-NP	x	O
move	NN	I-NP	x	O
,	,	O	,	O
spokesman	NN	B-NP	x	O
Tim	NNP	I-NP	Xx,	B-PER
Wagner	NNP	I-NP	Xx	I-PER
said	VBD	B-VP	x	O
.	,	O	.	O

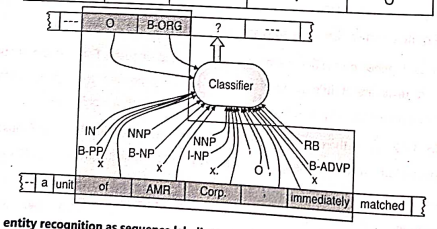


Fig. 4.2.3 : Named entity recognition as sequence labeling. The features available to the classifier during training and classification are those in the boxed area.

4.2.1(C) A Neural Algorithm for NER

- The standard neural algorithm for NER is based on the bi-LSTM. Recall that in that model, word and character embeddings are computed for input word w_i. These are passed through a left-to-right LSTM and a right-to-left LSTM, whose outputs are concatenated (or otherwise combined) to produce a single output layer at position i.
- In the simplest method, this layer can then be directly passed onto a softmax that creates a probability distribution over all NER tags, and the most likely tag is chosen as t_i. For named entity tagging this greedy approach to decoding is insufficient, since it doesn't allow us to impose the strong constraints neighboring tokens have on each other (e.g., the tag I-PER must follow another I-PER or B-PER).
- Instead a CRF layer is normally used on top of the bi-LSTM output, and the Viterbi decoding algorithm is used to decode. Fig. 4.2.4 shows a sketch of the algorithm.

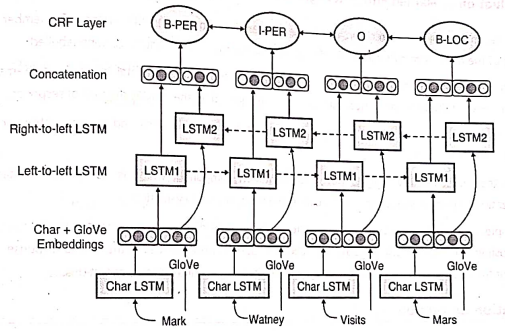


Fig. 4.2.4 : Putting it all together : character embeddings and words together in a bi-LSTM sequence model. After Lample et al. (2016b)

4.2.1(D) Rule-based NER

- While machine learned (neural or MEMM/CRF) sequence models are the norm in academic research, commercial approaches to NER are often based on pragmatic combinations of lists and rules, with some smaller amount of supervised machine learning (Chiticariu et al., 2013).
- For example IBM System T is a text understanding architecture in which a user specifies complex declarative constraints for tagging tasks in a formal query language that includes regular expressions, dictionaries, semantic constraints for tagging tasks in a formal query language that includes regular expressions, dictionaries, semantic constraints, NLP operators, and table structures, all of which the system compiles into an efficient extractor (Chiticariu et al., 2018).
- One common approach is to make repeated rule-based passes over a text, allowing the results of one pass to influence the next. The stages typically first involve the use of rules that have extremely high precision but low recall. Subsequent stages employ more error-prone statistical methods that take the output of the first pass into account.

1. First, use high-precision rules to tag unambiguous entity mentions.

2. Then, search for substring matches of the previously detected names.
3. Consult application-specific name lists to identify likely name entity mentions from the given domain.
4. Finally, apply probabilistic sequence labeling techniques that make use of the tags from previous stages as additional features.

- The intuition behind this staged approach is twofold. First, some of the entity mentions in a text will be more clearly indicative of a given entity's class than others.
- Second, once an unambiguous entity mention is introduced into a text, it is likely that subsequent shortened versions will refer to the same entity (and thus the same type of entity).

4.2.2 Evaluation of Named Entity Recognition

- The familiar metrics of recall, precision, and F1 measure are used to evaluate NER systems. Remember that recall is the ratio of the number of correctly labelled responses to the total that should have been labelled;
- Precision is the ratio of the number of correctly labelled responses to the total labelled; and F-measure is the harmonic mean of the two. For named entities, the entity rather than the word is the unit of response.
- Thus in the example in Table 4.2.6, the two entities Tim Wagner and AMR Corp. and the non-entity said would each count as a single response.
- The fact that named entity tagging has a segmentation component which is not present in tasks like text categorization or part-of-speech tagging causes some problems with evaluation.
- For example, a system that labelled *American* but not *American Airlines* as an organization would cause two errors, a false positive for O and a false negative for I-ORG. In addition, using entities as the unit of response but words as the unit of training means that there is a mismatch between the training and test conditions.

4.3 Relation Extraction

- Next on our list of tasks is to discern the relationships that exist among the detected entities. Let's return to our sample airline text :

  Citing high fuel prices, [ORG United Airlines] said [TIME Friday] it has increased fares by [MONEY $6] per round trip on flights to some cities also served by lower-cost carriers. [ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said. [ORG United], a unit of [ORG UAL Corp.], said the increase took effect [TIME Thursday] and applies to most routes where it competes against discount carriers, such as [LOC Chicago] to [LOC Dallas] and [LOC Denver] to [LOC San Francisco].

- The text tells us, for example, that Tim Wagner is a spokesman for American Airlines, that United is a unit of UAL Corp., and that American is a unit of AMR.
- These binary relations are instances of more generic relations such as part-of or employs that are fairly frequent in news-style texts.
- Fig. 4.3.1 lists the 17 relations used in the ACE relation extraction evaluations and Table 4.3.1 shows some sample relations. We might also extract more domain-specific relation such as the notion of an airline route.
- For example from this text we can conclude that United has routes to Chicago, Dallas, Denver, and San Francisco.

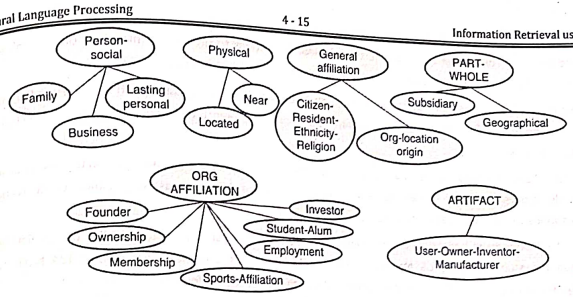


Fig. 4.3.1 : The 17 relations used in the ACE relation extraction task

Table 4.3.1 : Semantic relations with examples and the named entity types they involve

Relations	Types	Examples
Physical-located	PER-GPE	He was in Tennessee
Part-whole-subsidiary	ORG-ORG	XYZ, the parent company of ABC
Person-Social-Family	PER-PER	Yoko's husband John
ORG-AFF-Founder	PE-ORG	Sten Jobs, co-founder of apple...

Table 4.3.2 : A model-based view of the relations and entities in our sample text

Domain	D = {a, b, c, d, e, f, g, h, i}
United, UAL, American Airlines, AMR	a, b, c, d
Tim Wanger	e
Chicago, Dallas, Denver, and San Francisco	f, g, h, i
Classes	
United, UAL, American and AMR are organizations	Org = {a, b, c, d}
Time Wagner is person	Pers = {e}
Chicago, Dallas, Denver, and San Francisco	Loc = {f, g, h. i}
Relations	
United is a unit of UAL	PartOf = {⟨a, b⟩, ⟨c, d⟩}
America is a unit of AMR	
Tim Wagner works for American Airlines	OrgAff = {⟨c, e⟩}
United serves Chicago, Dallas, Denver, and San Francisco	Serves = {⟨a, f⟩, ⟨a, g⟩, ⟨a, h⟩, ⟨a, i⟩}

- These relations correspond nicely to the model-theoretic notions to ground the meanings of the logical forms. That is, a relation consists of a set of ordered tuples over elements of a domain.
- In most standard information extraction applications, the domain elements correspond to the named entities that occur in the text, to the underlying entities that result from co-reference resolution, or to entities selected from a domain ontology.
- Table 4.3.2 shows a model-based view of the set of entities and relations that can be extracted from our running example. Notice how this model-theoretic view subsumes the NER task as well; named entity recognition corresponds to the identification of a class of unary relations.
- Sets of relations have been defined for many other domains as well. For example UMLS, the Unified Medical Language System from the US National Library of Medicine has a network that defines 134 broad subject categories, entity types, and 54 relations between the entities, such as the following :

Entity	Relation	Entity
Injury	Disrupts	Physiological function
Bodily location	Location of	Biological function
Anatomical structure	Part of	Organism
Pharmacologic substance	Cause	Pathological function
Pharmacologic substance	Treats	Pathologic function

- Given a medical sentence like this one :

  Doppler echocardiography can be used to diagnose left anterior descending artery stenosis in patients with type 2 diabetes

- We could thus extract the UMLS relation :

  Echocardiography, Doppler Diagnoses Acquired stenosis

- Wikipedia also offers a large supply of relations, drawn from **infoboxes**, structured tables associated with certain Wikipedia articles. For example, the Wikipedia **infobox** for Stanford includes structured facts like state = "California" or president = "Mark Tessier-Lavigne". These facts can be turned into relations like president-of or located-in. or into relations in a metalanguage called **RDF** (Resource Description Framework). An **RDF triple** is a tuple of entity-relation-entity, called a subject-predicate-object expression. Here's a sample RDF triple :

subject	predicate	object
Golden Gate Park	location	San Francisco

- For example the crowdsourced DBpedia (Bizer et al., 2009) is an ontology derived from Wikipedia containing over 2 billion RDF triples. Another dataset from Wikipedia infoboxes, Freebase (Bollacker et al., 2008), now part of Wikidata (Vrande˘ci´c and Kr¨otzsch, 2014), has relations like:

  people/person/nationality

  location/location/contains

- WordNet or other ontologies offer useful ontological relations that express hierarchical relations between words or concepts. For example WordNet has the is-a or hypernym relation between classes,

  Giraffe is-a ruminant is-a ungulate is-a mammal is-a vertebrate ...

- WordNet also has *Instance-of* relation between individuals and classes, so that for example *San Francisco* is in the *Instance-of* relation with *city*. Extracting these relations is an important step in extending or building ontologies.
- There are five main classes of algorithms for relation extraction : **handwritten patterns, supervised machine learning, semi-supervised** (via **bootstrapping** and via **distant supervision**), and **unsupervised.** We'll introduce each of these in the next sections.

4.3.1 Using Patterns to Extract Relations

- The earliest and still common algorithm for relation extraction is lexico-syntactic patterns, first developed by Hearst (1992a). Consider the following sentence :

  Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.

- Hearst points out that most human readers will not know what *Gelidium* is, but that they can readily infer that it is a kind of (a **hyponym** of) *red algae*, whatever that is.
- She suggests that the following **lexico-syntactic pattern**

  NP_0 such as NP_1 { , NP_2..., (and|or) NP_i}, $i \geq 1$

- Implies the following semantics

  $\forall NP_i, i \geq 1$, hyponym (NP_i, NP_0)

  Allowing us to infer

  hyponym (Gelidium; red algae)

- Table 4.3.3 shows five patterns Hearst (1992a, 1998) suggested for inferring the hyponym relation; we've shown NPH as the parent/hyponym.
- Modern versions of the pattern-based approach extend it by adding named entity constraints. For example if our goal is to answer questions about "Who holds what office in which organization?", we can use patterns like the following :

Table 4.3.3 : Hand-built lexico-syntactic patterns for finding hypernyms, using fg to mark optionality

NP { , NP} * {,} (and|or) other NP_H	Temples, treasuries and other important civic buildings
NP_H such as {NP,} * {or|and} NP	Red algae such as Gelidium
such NP_H as (NP,) * {(or|and)} NP	Such authors s Herrick, Goldsmith, and Shakespeare
NP_H {,} including {NP,}* {(or|and)} NP	Common-law countries, including Canada and England
NP_H {,} especially {NP} * {(or|and)} NP	European countries, especially France, England and Spain

PER, POSITION of ORG :

- George Marshall, Secretary of State of the United States

- PER (named | appointed | chose | etc.) PER Prep? POSITION
- Truman appointed Marshall Secretary of State
- PER [be]? (named | appointed | etc.) Prep? ORG POSITION
- George Marshall was named US Secretary of State
- Hand-built patterns have the advantage of high-precision and they can be tailored to specific domains. On the other hand, they are often low-recall, and it's a lot of work to create them for all possible patterns.

4.3.2 Relation Extraction via Supervised Learning

- Supervised machine learning approaches to relation extraction follow a scheme that should be familiar by now. A fixed set of relations and entities is chosen, a training corpus is hand-annotated with the relations and entities, and the annotated texts are then used to train classifiers to annotate an unseen test set.
- The most straightforward approach has three steps, illustrated in Fig. 4.3.2. Step one is to find pairs of named entities (usually in the same sentence). In step two, a filtering classifier is trained to make a binary decision as to whether a given pair of named entities are related (by any relation).
- Positive examples are extracted directly from all relations in the annotated corpus, and negative examples are generated from within-sentence entity pairs that are not annotated with a relation.
- In step 3, a classifier is trained to assign a label to the relations that were found by step 2.
- The use of the filtering classifier can speed up the final classification and also allows the use of distinct feature-sets appropriate for each task.
- For each of the two classifiers, we can use any of the standard classification techniques (logistic regression, neural network, SVM, etc.)

```
function FINDRELATIONS(words) returns relations
    relations ← nil
    entities ← FINDENTITIES(words)
    forall entity pairs ⟨e1, e2⟩ in entities do
        if RELATED?(e1, e2)
            relations ← relations+CLASSIFYRELATION(e1, e2)
```

Fig. 4.3.2 : Finding and classifying the relations among entities in a text

- For the feature-based classifiers like logistic regression or random forests the most important step is to identify useful features. Let's consider features for classifying the relationship between American Airlines (Mention 1, or M1) and Tim Wagner (Mention 2, M2) from this sentence:
- American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said Useful word features include
  - The headwords of M1 and M2 and their concatenation

    Airlines Wagner Airlines-Wagner
  - Bag-of-words and bigrams in M1 and M2

    American, Airlines, Tim, Wagner, American Airlines, Tim Wagner

  - Words or bigrams in particular positions

    M2: -1 spokesman

    M2: +1 said
  - Bag of words or bigrams between M1 and M2:

    a, AMR, of, immediately, matched, move, spokesman, the, unit
  - Stemmed versions of the same

    Embeddings can be used to represent words in any of these features. Useful named entity features include
  - Named-entity types and their concatenation

    (M1: ORG, M2: PER, M1M2: ORG-PER)
  - Entity Level of M1 and M2 (from the set NAME, NOMINAL, PRONOUN)

    M1: NAME [it or he would be PRONOUN]

    M2: NAME [the company would be NOMINAL]
  - Number of entities between the arguments (in this case 1, for AMR)
- The syntactic structure of a sentence can also signal relationships among its entities. Syntax is often featured by using strings representing syntactic paths : the (dependency or constituency) path traversed through the tree in getting from one entity to the other.
  - Base syntactic chunk sequence from M1 to M2

    NP NP PP VP NP NP
  - Constituent paths between M1 and M2

    NP ↑ NP ↑ S ↑ S ↓ NP
  - Dependency-tree paths

    Airlines $\leftarrow_{sub\,j}$ matched $\leftarrow_{comp}$ said $\leftarrow_{sub\,j}$ Wagner
- Fig. 4.3.3 summarizes many of the features we have discussed that could be used for classifying the relationship between *American Airlines* and *Tim Wagner* from our example text.
- Neural models for relation extraction similarly treat the task as supervised classification. One option is to use a similar architecture as we saw for named entity tagging: a bi-LSTM model with word embeddings as inputs and a single softmax classification of the sentence output as a 1-of-N relation label.
- Because relations often hold between entities that are far part in a sentence (or across sentences), it may be possible to get higher performance from algorithms like convolutional nets (dos Santos et al., 2015) or chain or tree LSTMS (Miwa and Bansal 2016, Peng et al. 2017).
- In general, if the test set is similar enough to the training set, and if there is enough hand-labelled data, supervised relation extraction systems can get high accuracies.
- But labelling a large training set is extremely expensive and supervised models are brittle: they don't generalize well to different text genres.
- For this reason, much research in relation extraction has focused on the semi-supervised and unsupervised approaches we turn to next.

M1 headword	Airlines (as a word token or an embedding)
M2 headword	Wagner
Word(s) before M1	NONE
Word(s) after M2	said
Bag of words between	{a, unit, of, AMR, Inc., immediately, matched, the move, spokesman}
M1 type	ORG
M2 type	PERS
Concatenated types	ORG-PERS
Constituent path	NP ↑ NP ↑ S ↑ S ↑ NP
Base phrase path	NP → NP → PP → NP → VP → NP → NP
Typed dependency path	Airlines $\leftarrow_{sub\ j}$ matched $\leftarrow_{comp}$ said $\rightarrow_{sub\ j}$ Wagner

Fig. 4.3.3 : Sample of features extracted during classification of the <American Airlines, Tim Wagner> tuple; M1 is the first mention, M2 the second.

4.3.3 Semi-supervised Relation Extraction via Bootstrapping

- Supervised machine learning assumes that we have lots of labeled data. Unfortunately, this is expensive. But suppose we just have a few high-precision seed patterns, or perhaps a few seed tuples. That's enough to bootstrap a classifier! Bootstrapping proceeds by taking the entities in the seed pair, and then finding sentences (on the web, or whatever dataset we are using) that contain both entities.
- From all such sentences, we extract and generalize the context around the entities to learn new patterns. Fig. 4.3.4 sketches a basic algorithm.

function BOOTSTRAP(*Relation R*) **returns** *new relation tuples*

tuples ← Gather a set of seed tuples that have relation *R*
iterate
  sentences ← find sentences that contain entities in *tuples*
  patterns ← generalize the context between and around entities in *sentences*
  newpairs ← use *patterns* to grep for more tuples
  newpairs ← *newpairs* with high confidence
  tuples ← *tuples* + *newpairs*
return tuples

Fig. 4.3.4 : Bootstrapping from seed entity pairs to learn relations.

- Suppose, for example, that we need to create a list of airline/hub pairs, and we know only that Ryanair has a hub at Charleroi. We can use this seed fact to discover new patterns by finding other mentions of this relation in our corpus.
- We search for the terms Ryanair, Charleroi and hub in some proximity. Perhaps we find the following set of sentences :

- Budget airline Ryanair, which uses Charleroi as a hub, scrapped all weekend flights out of the airport.
- All flights in and out of Ryanair's Belgian hub at Charleroi airport were grounded on Friday...
- A spokesman at Charleroi, a main hub for Ryanair, estimated that 8000 passengers had already been affected.

- From these results, we can use the context of words between the entity mentions, the words before mention one, the word after mention two, and the named entity types of the two mentions, and perhaps other features, to extract general patterns such as the following :

/[ORG], which uses [LOC] as a hub /
/ [ORG]' hub at [LOC] /
/ [LOC] a main hub for [ORG] /

- These new patterns can then be used to search for additional tuples.

4.4 Reference Resolution

- In this section we will study problem of reference, the process by which REFERENCE speakers use expressions like John and he in passage (*John went to Bill's car dealership to check out an Acura Integra. He looked at it for about an hour.*) to denote a person named John.
- A natural language expression used to perform reference is called a referring expression, and the entity that is referred to is called the referent.
- Thus, John and he in passage (*John went to Bill's car dealership to check out an Acura Integra. He looked at it for about an hour.*) are referring expressions, and John is REFERENT their referent.
- Two referring expressions that are used to refer to the same entity are said to corefer, thus John and he are corefer.
- The discourse model contains representations of the entities that have been referred to in the discourse and the relationships in which they participate.
- Thus, there are two components required by a system to successfully produce and interpret referring expressions: a method for constructing a discourse model that evolves with the dynamically-changing discourse it represents, and a method for mapping between the signals that various referring expressions encode and the hearer's set of beliefs, the latter of which includes this discourse model.
- We will speak in terms of two fundamental operations to the discourse model.
- When a referent is first mentioned in a discourse, we say that a representation for it is evoked into the model.
- Natural languages provide speakers with a variety of ways to refer to entities. Say that your friend has an Acura Integra automobile and you want to refer to it.
- Depending on the operative **discourse context**, you might say *it, this, that, this car, that car, the car, the Acura, the Integra*, or *my friend's car*, among many other possibilities. However, you are not free to choose between any of these alternatives in any context.
- For instance, you cannot simply say it or the Acura if the hearer has no prior knowledge of your friend's car, it has not been mentioned before, and it is not in the immediate surroundings of the discourse participants (i.e., the **situational context** of the discourse).

- Upon subsequent mention, this representation is accessed from the model. The operations and relationships are illustrated in Fig. 4.4.1.

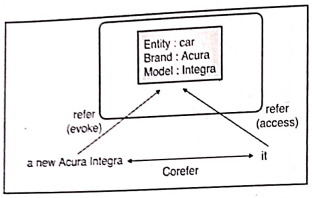


Fig. 4.4.1 : Reference operations and relationships

4.5 Co-reference Resolution

- Coreference resolution is the task of clustering mentions in text that refer to the same underlying real world entities. After finding and grouping these mentions we can resolve them by replacing, as stated above, pronouns with noun phrases. It is an important step for a lot of higher level NLP tasks that involve natural language understanding such as document summarization, question answering, and information extraction.

```
                +-----------+
                |           |
I voted for Obama because he was most aligned with my values", she said.
 |                                          |           |
 +------------------------------------------+-----------+
```

- "I", "my", and "she" belong to the same cluster and "Obama" and "he" belong to the same cluster.

Reference Phenomena

- The set of referential phenomena that natural languages provide is quite rich indeed.
- In this section, we provide a brief description of several basic reference phenomena.
- We first survey five types of referring expression: indefinite noun phrases, definite noun phrases, pronouns, demonstratives, and one-anaphora.
- We then describe three types of referents that complicate the reference resolution problem: inferrables, discontinuous sets, and generics.

Indefinite Noun Phrases

- Indefinite reference introduces entities that are new to the hearer into the discourse context.
- The most common form of indefinite reference is marked with the determiner a (or an), as in (1), but it can also be marked by a quantifier such as some (2) or even the determiner this (3).

  (1) I saw an Acura Integra today.

  (2) Some Acura Integras were being unloaded at the local dealership today.

  (3) I saw this awesome Acura Integra today.

- Such noun phrases evoke a representation for a new entity that satisfies the given description into the discourse model.

Definite Noun Phrases :

- Definite reference is used to refer to an entity that is identifiable to the hearer, either because it has already been mentioned in the discourse context (and thus is represented in the discourse model), it is contained in the hearer's set of beliefs about the world, or the uniqueness of the object is implied by the description itself.
- The case in which the referent is identifiable from discourse context is shown in (4).

  (4) I saw an Acura Integra today. The Integra was white and needed to be washed.

- Definite noun phrase reference requires that an entity be accessed from either the discourse model or the hearer's set of beliefs about the world.

Pronouns :

- Another form of definite reference is pro nominalization, illustrated in example (5).

  (5) I saw an Acura Integra today. It was white and needed to be washed.

- The constraints on using pronominal reference are stronger than for full definite noun phrases, requiring that the referent have a high degree of activation or salience in the discourse model.
- Pronouns usually (but not always) refer to entities that were introduced no further than one or two sentences back in the ongoing discourse, whereas definite noun phrases can often refer further back.
- This is illustrated by the difference between sentences (6d) and (6d').

  (6d) ?? He also said that he bought it yesterday.

  (6d') He also said that he bought the Acura yesterday.

Demonstratives :

- Demonstrative pronouns, like this and that, behave somewhat differently that simple definite pronouns like it.
- They can appear either alone or as determiners, for instance, this Acura, that Acura.
- The choice between two demonstratives is generally associated with some notion of spatial proximity: this indicating closeness and that signaling distance.
- Spatial distance might be measured with respect to the discourse participants' situational context, as in (7).

  (7) [John shows Bob an Acura Integra and a Mazda Miata] Bob (pointing): I like this better than that.

One Anaphora :

- One-anaphora, exemplified in (8), blends properties of definite and indefinite reference

  (8) I saw no less than 6 Acura Integrals today. Now I want one.

- This use of **one** can be roughly paraphrased by **one** of **them**, in which them refers to a plural referent (or generic one, as in the case of (8), and **one** selects a member from this set.
- Thus, **one** may evoke a new entity into the discourse model, but it is necessarily dependent on an existing referent for the description of this new entity.

4.6 Cross-Lingual Information Retrieval

- The purpose of cross-lingual Information Retrieval (CLIR) is to support queries in one language with a collection in other languages. The rapid growth and availability of online information in many languages have made the task of finding relevant information from large, multilingual text collections a field of active research in the IR and NLP communities.

- In many respects, CLIR shares the same goals as machine retrieval. Well- known IR techniques, such as vector space retrieval, latent semantic indexing, similarity measures for matching documents and query processing procedures, are equally useful in CLIR.
- However, CLIR differs from IR in several significant ways. Most importantly, IR requires no translation phase. Since the queries and documents are in different languages, CLIR requires a machine translation phase along with the usual monolingual retrieval phase. For this purpose. CLIR adopts various techniques explored in NLP research.
- It seems that CLIR is simply a matter of coupling IR and MT. However, it is not so. The special nature of CLIR places constraints on the input to MT, which makes a straightforward coupling unfeasible. The central challenge in CLIR is to find the most effective way to bridge the language barrier between queries and documents.
- Methods of CLIR are typically divided into approaches based on the use of bilingual dictionaries or MT; on corpuses and a range of IR- specific statistical measures; and concept-driven approaches.

Issues :

Query processing : A query is a statement of information need from the user. There are three main problems associated with the processing of query terms for CLIR.

- How can a query term in LI be expressed in L2?
- What mechanisms determine which of the possible translations of text from L1 to L2 should be retained?
- In cases where more than one translation is retained, how are the different translation alternatives to be weighted?

Indexing :

- Choosing good terms and weighting them is hard enough in one language, but when the problem is extended to documents in multiple languages, it becomes considerably more difficult.
- Standard IR methods for indexing involve a small amount of language-specific processing. Multilingual document preparation and pre-processing techniques have been the focus of much recent research.
- This includes tokenizing (separating the continuous character stream into individual words), part-of-speech tagging, stemming and demorphing (for example, converting inflected words into their root forms plus associated information, a task that can be quite complex in highly inflected languages such as Arabic).
- In addition, new techniques for extracting collocational and phrasal information, both monolingually and multilingually, are being developed.

Matching and ranking :

The matching problem is primarily difficult in CLIR due to imperfect correspondence between languages. Similarity metrics based on keyword matching may create problems due to conceptual mismatches.

4.6.1 Approaches to CLIR

- Fig. 4.6.1 shows the general approach to CLIR. The user submits a query in one language and the retrieval system responds by returning documents in a language different from the query language.
- As the query language is different from the language of the documents in the collection, it has to be translated before the actual retrieval. The various approaches that can be used for CLIR are as follows.

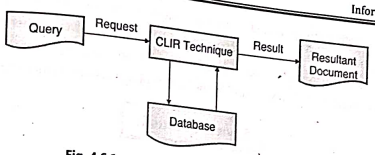


Fig. 4.6.1 : Cross-lingual information retrieval

1. **Dictionary-based Query Translation :**

Query keywords are translated to the target language using machine readable dictionaries (MRD). The main problems associated with this approach are phrase identification and translation, source ambiguity, translation ambiguity, the coverage of the dictionary and processing of inflected words (Pirkola et al. 2004, Helhund et al. 2001).

2. **Machine Translation**

- In CLIR, MT can be implemented in two ways. The first is to use an MT system to translate foreign language documents into the language of the user's query. The second way is to translate the user's query into the target language (the language of the documents in the stored collection). The target language query is then used to retrieve target language documents using classical IR techniques.
- Both methods require an MT stage that is separate from the retrieval stage. An ambiguity problem exists in the MT component, since the translated query.
- MT query does not necessarily represent the sense of the original systems normally attempt to determine the correct word sense for translation by using context analysis (Braschler 2004). However, a typical search engine query is small and lacks context.

3. **Corpus-based**

- A corpus is a repository for a collection of natural language material, such as text, paragraphs, and sentences from one or many languages. Two types of corpuses have been used in query translation: parallel and comparable. Parallel corpuses consist of the same text in more than one language.
- When retrieving text from a parallel corpus, the source language query need not be translated. The source language query is used to match against the source language component of the corpus, and the target language component aligned to it is retrieved.
- A number of alignment methods have been proposed in literature (Carpuat et al. 2006). Comparable corpuses contain text in more than one language.
- The texts in each language are not translations of each other, but cover the same topic area, and hence contain equivalent vocabulary.

4. **Concept-driven :**

- Concept-driven approaches exploit multilingual ontologies or thesauri to bridge the gap between surface linguistic forms and meanings. The terms and phrases from multiple languages that refer to the same concept are mapped into a language-independent scheme.

- In this approach, both documents and queries are mapped into the conceptual interlingua, which, permits matching and retrieval based on any combination of languages involved, rather than relying on pair-wise translations. This seems particularly appropriate for domains (and languages) for which extensive multilingual ontologies are available, such as UMLS (unified medical language system) in the medical domain.

Review Questions

Q.1 Define Information Retrieval. Explain IR process with suitable diagram.

Q.2 Explain Term-Weighting with suitable example.

Q.3 What is Relation extraction ?

Q.4 Explain classes of algorithms for relation extraction.

Q.5 What are the referring expression? Explain each with suitable example.

Q.6 What are the approaches to cross lingual Information retrival.

□□□

5

Unit - V

NLP Tools and Techniques

Syllabus

Prominent NLP Libraries : Natural Language Tool Kit (NLTK), spaCy, TextBlob, Gensim etc. Linguistic Resources: Lexical Knowledge Networks, WordNets, Indian Language WordNet (IndoWordnet), VerbNets, PropBank, Treebanks, Universal Dependency Treebanks Word Sense Disambiguation: Lesk Algorithm Walker's algorithm, WordNets for Word Sense Disambiguation.

5.1 Prominent NLP Libraries

5.1.1 Natural Language Tool Kit (NLTK)

- The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical Natural Language Processing (NLP).
- It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. It also includes graphical demonstrations and sample data sets as well as accompanied by a cook book and a book which explains the principles behind the underlying language processing tasks that NLTK supports.
- NLTK defines an infrastructure that can be used to build NLP programs in Python. It provides basic classes for representing data relevant to natural language processing; standard interfaces for performing tasks such as part-of-speech tagging, syntactic parsing, and text classification; and standard implementations for each task that can be combined to solve complex problems.
- NLTK was originally created in 2001 as part of a computational linguistics course in the Department of Computer and Information Science at the University of Pennsylvania.
- Since then it has been developed and expanded with the help of dozens of contributors. It has now been adopted in courses in dozens of universities, and serves as the basis of many research projects. Table 5.1.1 lists the most important NLTK modules.

Table 5.1.1 : Language processing tasks and corresponding NLTK modules with examples of functionality

Language processing task	NLTK modules	Functionality
Accessing corpora	nltk. corpus	Standardized interfaces to corpora and lexicons
String processing	nltk.tokenize, nltk.stem	Tokenizers, sentence tokenizers, stemmers
Collocation discovery	nltk.collocations	t-test, chi-squared, point-wise mutual information
Part-of-speech tagging	nltk.tag	n-gram, backoff, Brill, HMM, TnT

Language processing task	NLTK modules	Functionality
Classification	nltk.classify, nlkt.cluster	Decision tree, maximum entropy, naïve Bayes, EM, k-means
Chunking	nltk.Chunk	Regular expression , n-gram, named entity
Parsing	nltk.parse	Chart, feature-based, unification, probabilistic, dependency
Semantic interpretation	nltk.sem., nltk.inference	Lambada calculus, first order logic, model checking
Evaluation metrics	nltk.metrics	Precision, recall, agreement coefficients
Probability and estimating	nltk.probability	Frequency distributions, smoothed probability distributions
Applications	nltk.app, nltk.chat	Graphical concordancer, parsers, wordNet Browser, chatbots

- NLTK was designed with four primary goals in mind :
  - **Simplicity :** To provide an intuitive framework along with substantial building blocks, giving users a practical knowledge of NLP without getting bogged down in the tedious house-keeping usually associated with processing annotated language data
  - **Consistency :** To provide a uniform framework with consistent interfaces and data structures, and easily guessable method names.
  - **Extensibility :** To provide a structure into which new software modules can be easily accommodated, including alternative implementations and competing approaches to the same task
  - **Modularity :** To provide components that can be used independently without needing to understand the rest of the toolkit
- Contrasting with these goals are three non-requirements potentially useful qualities that we have deliberately avoided. First, while the toolkit provides a wide range of functions, it is not encyclopedic; it is a toolkit, not a system, and it will continue to evolve with the field of NLP.
- Second, while the toolkit is efficient enough to support meaningful tasks, it is not highly optimized for runtime performance; such optimization soften involve more complex algorithms, or implementations in lower-level programming languages such as C or C++.
- This would make the software less readable and more difficult to install. Third, we have tried to avoid clever programming tricks, since we believe that clear implementations are preferable to ingenious yet indecipherable ones.
- NLTK includes more than 50 corpora and lexical sources such as the Penn Treebank Corpus, Open Multilingual Wordnet, Problem Report Corpus, and Lin's Dependency Thesaurus.

Installing NLTK :

- NLTK requires Python versions 3.7, 3.8, 3.9, 3.10 or 3.11.
- For Windows users, it is strongly recommended that you go through this guide to install Python 3 successfully **https://docs.python-guide.org/starting/install3/win/#install3-windows**

- Setting up a Python Environment (Mac/Unix/Windows)
- Please go through this guide to learn how to manage your virtual environment managers before you install NLTK, **https://docs.python-guide.org/dev/virtualenvs/**
- Alternatively, you can use the Anaconda distribution installer that comes "batteries included" **https://www.anaconda.com/distribution/Mac/Unix**
- **Install NLTK :** run pip install --user -U nltk
- **Install Numpy (optional) :** run pip install --user -U numpy
- **Test installation :** run python then type import nltk
- For older versions of Python it might be necessary to install setuptools (see https://pypi.python.org/pypi/setuptools) and to install pip (sudoeasy_install pip).

Windows :

- These instructions assume that you do not already have Python installed on your machine. 32-bit binary installation

  Install Python 3.8 : https://www.python.org/downloads/ (avoid the 64-bit versions)

  Install Numpy (optional) : https://numpy.org/install/

  Install NLTK : https://pypi.python.org/pypi/nltk

  Test installation : Start>Python38, then type import nltk

Installing NLTK Data

- After installing the NLTK package, please do install the necessary datasets/models for specific functions to work.
- If you're unsure of which datasets/models you'll need, you can install the "popular" subset of NLTK data, on the command line type **python -m nltk.downloader** popular, or in the Python interpreter **import nltk; nltk.download('popular')**

5.1.2 spaCy

- SpaCy is a free, open-source library for NLP in Python written in Cython. spaCy is designed to make it easy to build systems for information extraction or general-purpose natural language processing.
- spaCy provides a variety of linguistic annotations to give you **insights into a text's grammatical structure.** This includes the word types, like the parts of speech, and how the words are related to each other. For example, if you are analyzing text, it makes a huge difference whether a noun is the subject of a sentence, or the object or whether "google" is used as a verb, or refers to the website or company in a specific context.
- spaCy is designed specifically for **production use** and helps you build applications that process and "understand" large volumes of text. It can be used to build **information extraction** or **natural language understanding** systems, or to pre-process text for **deep learning**.
- You can install spaCy using **pip,** a Python package manager. It's a good idea to use a **virtual environment** to avoid depending on system-wide packages.

1. spaCy's Statistical Models
2. spaCy's Processing Pipeline

1. **spaCy's Statistical Models :**
   - These models are the power engines of spaCy. These models enable spaCy to perform several NLP related tasks, such as part-of-speech tagging, named entity recognition, and dependency parsing.
   - Below the different statistical models in spaCy along with their specifications :
     - **en_core_web_sm :** English multi-task CNN trained on OntoNotes. Size – 11 MB.
     - **en_core_web_md :** English multi-task CNN trained on OntoNotes, with GloVe vectors trained on Common Crawl. Size – 91 MB.
     - **en_core_web_lg :** English multi-task CNN trained on OntoNotes, with GloVe vectors trained on Common Crawl. Size – 789 MB.
   - Importing these models is super easy. We can import a model by just executing *spacy.load('model_name')* as shown below :

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

2. **spaCy's Processing Pipeline**
   - The first step for a text string, when working with spaCy, is to pass it to an **NLP object.** This object is essentially a pipeline of several text pre-processing operations through which the input text string has to go through.

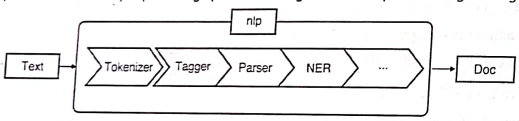


Fig. 5.1.1 : SpaCy Processing Pipeline

   - As you can see in the Fig. 5.1.1, the NLP pipeline has multiple components, such as **tokenizer, tagger, parser, ner,** etc. So, the input text string has to go through all these components before we can work on it.

5.1.3 TextBlob

TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

Features :

- *Noun phrase extraction*
- *Part-of-speech tagging*
- *Sentiment analysis*
- *Classification (Naive Bayes, Decision Tree)*
- *Tokenization (splitting text into words and sentences)*
- *Word and phrase frequencies*
- *Parsing*

- *n-grams*
- *Word inflection (pluralization and singularization) and lemmatization*
- *Spelling correction*
- *Add new models or languages through extensions*
- *WordNet integration*

Installation

- *$ pip install –U textblob*
- *$ python –m textblob.download_corpora*

Dependencies

TextBlob depends on NLTK 3. NLTK will be installed automatically when you run *pip install textblob or python setup.py install.*

How to create TextBlob

First import

```
>>> from textblob import TextBlob
```

1. **Part-of-speech Tagging**

Part-of-speech tags can be accessed through the tags property.

```
>>>wiki.tags
[('Python', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('high-level', 'JJ'), ('general-purpose', 'JJ'), ('programming', 'NN'), ('language', 'NN')]
```

2. **Noun Phrase Extraction**

Similarly, noun phrases are accessed through the noun_phrases property.

```
>>>wiki.noun_phrases
WordList(['python'])
```

3. **Sentiment Analysis**

The sentiment property returns a named tuple of the form Sentiment(polarity, subjectivity). The polarity score is a float within the range [-1.0, 1.0]. The subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

```
>>> testimonial = TextBlob("Textblob is amazingly simple to use. What great fun!")
>>>testimonial.sentiment
Sentiment(polarity=0.39166666666666666, subjectivity=0.4357142857142857)
>>>testimonial.sentiment.polarity
0.39166666666666666
```

4. Tokenization

- You can break TextBlobs into words or sentences.

```
>>>zen = TextBlob("Beautiful is better than ugly. "
                 "Explicit is better than implicit. "
                 "Simple is better than complex.")
>>>zen.words
```

- WordList(['Beautiful', 'is', 'better', 'than', 'ugly', 'Explicit', 'is', 'better', 'than', 'implicit', 'Simple', 'is', 'better', 'than', 'complex'])

```
>>>zen.sentences
```

[Sentence("Beautiful is better than ugly."), Sentence("Explicit is better than implicit."), Sentence("Simple is better than complex.")]

- Sentence objects have the same properties and methods as TextBlobs.

```
>>> for sentence in zen.sentences :
    print(sentence.sentiment)
```

5. Words Inflection and Lemmatization :

- Each word in TextBlob.words or Sentence.words is a Word object (a subclass of unicode) with useful methods, e.g. for word inflection.

```
>>> sentence = TextBlob('Use 4 spaces per indentation level.')
>>>sentence.words
```

WordList(['Use', '4', 'spaces', 'per', 'indentation', 'level'])

```
>>>sentence.words[2].singularize()
```

'space'

```
>>>sentence.words[-1].pluralize()
```

'levels'

- Words can be lemmatized by calling the lemmatize method.

```
>>> from textblob import Word
>>> w = Word("octopi")
>>>w.lemmatize()
```

'octopus'

```
>>> w = Word("went")
>>>w.lemmatize("v")  # Pass in WordNet part of speech (verb)
```

'go'

6. WordNet Integration :

- You can access the synsets for a Word via the synsets property or the get_synsets method, optionally passing in a part of speech.

```
>>> from textblob import Word
>>> from textblob.wordnet import VERB
>>> word = Word("octopus")
>>>word.synsets
```

[Synset('octopus.n.01'), Synset('octopus.n.02')]

```
>>> Word("hack").get_synsets(pos=VERB)
```

[Synset('chop.v.05'), Synset('hack.v.02'), Synset('hack.v.03'), Synset('hack.v.04'), Synset('hack.v.05'), Synset('hack.v.06'), Synset('hack.v.07'), Synset('hack.v.08')]

- You can access the definitions for each synset via the definitions property or the define() method, which can also take an optional part-of-speech argument.

```
>>> Word("octopus").definitions
```

['tentacles of octopus prepared as food', 'bottom-living cephalopod having a soft oval body with eight long tentacles']

- You can also create synsets directly.

```
>>> from textblob.wordnet import Synset
>>> octopus = Synset('octopus.n.02')
>>> shrimp = Synset('shrimp.n.03')
>>>octopus.path_similarity(shrimp)
```

0.1111111111111111

5.1.4 Gensim

- ***Gensim = "Generate Similar"*** is a popular open source Natural Language Processing (NLP) library used for unsupervised topic modeling. It uses top academic models and modern statistical machine learning to perform various complex tasks such as :
  - Building document or word vectors
  - Corpora
  - Performing topic identification
  - Performing document comparison (retrieving semantically similar documents)
  - Analysing plain-text documents for semantic structure.
- Apart from performing the above complex tasks, Gensim, implemented in Python and Cython, is designed to handle large text collections using data streaming as well as incremental online algorithms. This makes it different from those machine learning software packages that target only in-memory processing.
- Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora. Target audience is the Natural Language Processing (NLP) and Information Retrieval (IR) community.

- In order to work on text documents, Gensim requires the words (aka tokens) be converted to unique ids. In order to achieve that, Gensim lets you create a Dictionary object that maps each word to a unique id.

Features :

- All algorithms are **memory-independent** w.r.t. the corpus size (can process input larger than RAM, streamed, out-of-core)
- **Intuitive interfaces**
  - Easy to plug in your own input corpus/data stream (simple streaming API)
  - Easy to extend with other Vector Space algorithms (simple transformation API)
- Efficient multicore implementations of popular algorithms, such as online **Latent Semantic Analysis (LSA/LSI/SVD), Latent Dirichlet Allocation (LDA), Random Projections (RP), Hierarchical Dirichlet Process (HDP) or word2vec deep learning.**
- **Distributed computing :** can run **Latent Semantic Analysis** and **Latent Dirichlet Allocation** on a cluster of computers.

Installation :

- This software depends on **NumPy and Scipy**, two Python packages for scientific computing. You must have them installed prior to installing **gensim.**
- It is also recommended you install a fast BLAS library before installing NumPy. This is optional, but using an optimized BLAS such as MKL, **ATLAS** or **OpenBLAS** is known to improve performance by as much as an order of magnitude. On OSX, NumPy picks up its vecLib BLAS automatically, so you don't need to do anything special.
- **Install the latest version of gensim :** pip install - upgrade gensim Or, if you have instead downloaded and unzipped the source tar.gz package: python setup.py install
- Gensim is being **continuously tested** under all **supported Python versions**. Support for Python 2.7 was dropped in gensim 4.0.0 – install gensim 3.8.3 if you must use Python 2.7.
- Many scientific algorithms can be expressed in terms of large matrix operations (see the BLAS note above). Gensim taps into these low-level BLAS libraries, by means of its dependency on NumPy. So while gensim-the-top-level-code is pure Python, it actually executes highly optimized Fortran/C under the hood, including multithreading (if your BLAS is so configured).
- Memory-wise, gensim makes heavy use of Python's built-in generators and iterators for streamed data processing. Memory efficiency was one of gensim's design goals, and is a central feature of gensim, rather than something bolted on as an afterthought.

5.2 Linguistic Resources

5.2.1 Lexical Knowledge Networks

- Lexical Knowledge Networks are resources which are critical for most NLP applications. This is evidenced by the success of the endeavour on WordNets which are now regarded as indispensable for work on semantic search, knowledge based machine translation, summarization, question answering and such other high end NLP applications.

- Lexico-semantic networks such as the Princeton WordNet are now considered as a vital resource.
- Competing lexical networks, such as ConceptNet, Hownet, MindNet, VerbNet, and FrameNet are also emerging as alternatives to WordNets.

5.2.2 Wordnets

- The most commonly used resource for sense relations in English and many other WordNet languages is the WordNet lexical database (Fellbaum,1998). English WordNet consists of three separate databases, one each for nouns and verbs and a third for adjectives and adverbs; closed class words are not included. Each database contains asset of lemmas, each one annotated with asset of senses.
- The widespread use of lexical relations in linguistic, psycholinguistic, and computational research has led to a number of efforts to create large electronic databases of such relations.
- These efforts have, in general, followed one of two basic approaches: mining information from existing dictionaries and thesauri, and handcrafting a database from scratch. Despite the obvious advantages of reusing existing resources, WordNet, the most well-developed and widely used lexical database for English, was developed using the latter approach (Beckwith et al., 1991).
- WordNet consists of three separate databases, one each for nouns and verbs, and a third for adjectives and adverbs; closed class lexical items are not included in WordNet. Each of the three databases consists of a set of lexical entries corresponding to unique orthographic forms, accompanied by sets of senses associated with each form.
- The WordNet3.0 release has 117,798 nouns, 11,529 verbs, 22,479 adjectives, and 4,481 adverbs. The average noun has 1.23 senses, and the average verb has2.16 senses. WordNet can be accessed on the Web or downloaded locally. Fig. 5.2.1 shows the lemma entry for the noun and adjective bass.

The noun "bass" has 8 senses in WordNet.

1. $bass^1$: (the lowest part of the musical range)
2. $bass^2$: bass $part^1$ – (the lowest part in polyphonic music)
3. $bass^3$: $basso^1$ – (an adult male singer with the lowest voice)
4. sea $bass^1$, $bass^4$ – (the lean flesh of a saltwater fish of the family serranidae)
5. freshwater $bass^1$, $bass^5$ – (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
6. $bass^6$, bass $voice^1$, $basso^2$ – (The lowest adult male singing voice)
7. $bass^7$: (the member with the lowest range of a family of musical instruments)
8. $bass^8$: (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Fig. 5.2.1 : A portion of the wordNet 3.0 entry for the noun bass

Gloss :

- A dictionary can provide useful information about the contexts related to the word senses (called glosses).
- We need to consider the alternative ways that dictionaries and thesauruses offer for defining senses. One is based on the fact that dictionaries or thesauruses give textual definitions for each sense called glosses. Here are the glosses for two senses of bank :
  1. Financial institution that accepts deposits and channels the money into lending activities
  2. Sloping land (especially the slope beside a body of water)

Synset :

- A synset is asset of synonyms that define a concept or word meaning About half of the synsets (~54%) contains only one term, about one third (~29%) 2 terms, about 10% 3 terms.
- An annotation (gloss) explaining the meaning is associated to each synset (especially to those containing a single term).

  A synset contains ~1.75 terms in average)

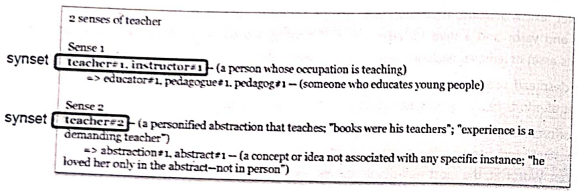


Fig. 5.2.2

- Glosses are properties of a synset, so that each sense included in the synset has the same gloss and can express this concept. Because they share glosses, synsets like this one are the fundamental unit associated with WordNet entries, and hence it is synsets, not word forms, lemmas, or individual senses, that participate in most of the lexical sense relations in WordNet.
- WordNet also labels each synset with a lexicographic category drawn from a semantic field for example the 26 categories for nouns shown in Fig. 5.2.3, as well as 15 for verbs (plus 2 for adjectives and 1 for adverbs). These categories are often called super senses, because they act as coarse semantic categories or groupings of senses which can be useful when word senses are too fine-grained (Ciaramita and Johnson 2003, Ciaramita and Altun 2006). Super senses have also been defined for adjectives (Tsvetkov et al., 2014) and prepositions (Schneider et al., 2018).

Category	Example	Category	Example	Category	Example
ACT	Service	GROUP	place	PLANT	tree
ANIMAL	dog	LOCATION	area	POSSESSION	price
ARTIFACT	car	MOTIVE	reason	PROCESS	process
ATTRIBUTE	quality	NATURAL EVENT	Experience	QUANTITY	amount
BODY	hair	NATURAL OBJECT	Flower	RELATION	portion
COGNITION	way	OTHER	stuff	SHAPE	square
COMMUNICATION	review	PERSON	people	STATE	pain
FEELING	Discomfort	PHENOMENON	result	SUBSTANCE	oil
FOOD	food			TIME	day

Fig 5.2.3 Super senses : 26 lexicographic categories for nouns in WordNet.

5.2.3 IndoWordNet

- IndoWordNet (Bhattacharyya, 2010) is as lexical database for various Indian languages, in which Hindi WordNet is the root and all other Indian language WordNets are linked through the expansion approach. Words and it's concepts are stored in a structure called the Lexical Matrix, where rows represent word meanings and columns represents the forms.
- IndoWordNet stores different words and relations mainly Lexical Relations and Semantic Relations. Different types of Lexical Relations such as Gradation for state, size, light, gender, temperature, color, time, quality, action, manner, Antonymy for action, amount, direction, gender, personality, place, quality, size, state, time, color, manner, Compound for nouns and Conjunction for verbs.
- Semantic Relation types such as Hypernymy for noun and verbs, Holonymy for nouns, Meronymy for component object, member collection, feature, activity, place, area, face, state, portion, mass, resource, process, position, area, Troponymy for verbs, Similar Attribute between noun and adjective, Function verb between noun and verb, Ability verb between noun and verb, Capability verb between noun and verb, Adverb modifies verb between adverb and verb, Causative for verb, Entailment for verb, Near synset and Adjective modifies noun between adjective and noun.
- **IndoWordNet** is a linked WordNet of major Indian languages, viz, Assamese, Bangla, Bodo, Gujarati, Hindi, Kannada, Kashmiri, Konkani, Malayalam, Manipuri, Marathi, Nepali, Oriya, Punjabi, Sanskrit, Tamil, Telugu and Urdu. These WordNets have been created using the expansion approach from Hindi WordNet and English.
- Each entry in the IndoWordNet consists of the following elements along with a related lexion in other Indian Languages, including English :
  1. Synonymy
  2. Gloss
  3. Example Sentence

5.2.4 VerbNet

- **VerbNet** (Kipper-Schuler, 2005) is a lexicon of verbs, and it includes thirty "core" thematic roles played by arguments to these verbs. Here are some example roles, accompanied by their definitions from the VerbNet Guidelines.3
- **AGENT :** "ACTOR in an event who initiates and carries out the event intentionally or consciously, and who exists independently of the event."
- **PATIENT :** "UNDERGOER in an event that experiences a change of state, location or condition, that is causally involved or directly affected by other participants, and exists independently of the event."
- **RECIPIENT :** "DESTINATION that is animate"
- **THEME :** "UNDERGOER that is central to an event or state that does not have control over the way the event occurs, is not structurally changed by the event, and/or is characterized as being in a certain position or condition throughout the state."
- **TOPIC :** "THEME characterized by information content transferred to another participant."

- VerbNet roles are organized in a hierarchy, so that a TOPIC is a type of THEME, which in turn is a type of UNDERGOER, which is a type of PARTICIPANT, the top-level category.
- In addition, VerbNet organizes verb senses into a class hierarchy, in which verb senses that have similar meanings are grouped together. Multiple meanings of the same word are called senses, and that WordNet identifies senses for many English words. VerbNet builds on WordNet, so that verb classes are identified by the WordNet senses of the verbs that they contain. For example, the verb class,

Asha gives Boyang a book :

- A first-order logical representation of this sentence is,

  ∃x : BOOK(x) ^GIVE(ASHA; BOYANG; x)
- Includes the first WordNet sense of loan and the second WordNet sense of lend.
- Each VerbNet class or subclass takes a set of thematic roles. For example, above expression(Asha gives Boyang a book) takes arguments with the thematic roles of AGENT, THEME, and RECIPIENT; the predicate TEACH takes arguments with the thematic roles AGENT, TOPIC, RECIPIENT, andSOURCE.5 So according to VerbNet, Asha and Boyang play the roles of AGENT and RECIPIENT in the sentences,

  a. Asha gave Boyang a book.

  b. Asha taught Boyang algebra.
- The book and algebra are both THEMES, but algebra is a subcategory of THEME - a TOPIC - because it consists of information content that is given to the receiver.

5.2.5 PropBank

- The **Proposition Bank,** or PropBank (Palmer et al., 2005), builds on this basic agent patient distinction, as a middle ground between generic thematic roles and roles that are specific to each predicate. Each verb is linked to a list of numbered arguments, with ARG0 as the proto-agent and ARG1 as the proto-patient. Additional numbered arguments are verb-specific. For example, for the predicate TEACH, the arguments are :
  - ARG0: the teacher
  - ARG1: the subject
  - ARG2: the student(s)
- Verbs may have any number of arguments: for example, WANT and GET have five, while EAT has only ARG0 and ARG1. In addition to the semantic arguments found in the frame files, roughly a dozen general-purpose adjuncts may be used in combination with any verb. These are shown in Table 5.2.1.
- PropBank-style semantic role labeling is annotated over the entire Penn Treebank. This annotation includes the sense of each verbal predicate, as well as the argument spans.
- The PropBank Corpus provides predicate-argument annotation for the entire Penn Treebank. Each verb in the treebank is annotated by a single instance in PropBank, containing information about the location of the verb, and the location and identity of its arguments :
- Each PropBank instance defines the following member variables :
  - Location information : *fileid, sentnum, wordnum*

  - Annotator information : *tagger*
  - Inflection information : *inflection*
  - Roleset identifier : *roleset*
  - Verb (aka predicate) location : *predicate*
  - Argument locations and types : *arguments*

Table 5.2.1: PropBank adjuncts (Palmer et al., 2005), sorted by frequency in the corpus

TMP	Time	Boyang ate a bagel [AM-TMP yesterday]
LOC	location	Asha studies in [AM-LOC Stuttgart]
MOD	modal verb	Asha [AM-MODwill] study in stuttgrat
ADV	general purpose	[AM-ADVluckily], Asha knew algebra
MNR	Manner	Asha ate [AM-MNR aggressively]
DIS	Discourse connective	[AM-DIS however), Asha prefers algebra
PRP	Purpose	Barry studied [AM-PRP to pass the bar]
DIR	Direction	Workers dumped burlap sacks [AM-DIR into a bin]
NEG	Negation	Asha does [AM-NEG not] speak Albanian
EXT	Extent	Prices increased [AM-EXT 4%]
CAU	Cause	Boyang returned the book [AM-CAU because it was overdue]

5.2.6 TreeBanks

- Sufficiently robust grammars consisting of context-free grammar rules can be used to assign a parse tree to any sentence. This means that it is possible to build a corpus where every sentence in the collection is paired with a corresponding parse tree. Such a syntactically annotated corpus is called a treebank. Treebanks play an important role in parsing, as well as in linguistic investigations of syntactic phenomena.
- A wide variety of treebanks have been created, generally through the use of parsers to automatically parse each sentence, followed by the use of humans (linguists) to hand-correct the parses. The Penn Treebank project has produced treebanks from the Brown, Switchboard, ATIS, and Wall Street Journal corpora of English, as well as treebanks in Arabic and Chinese. A number of treebanks use the dependency representation, including many that are part of the Universal Dependencies project (Nivre et al., 2016b).

Example: The Penn Treebank Project

- Fig. 5.2.4 shows sentences from the Brown and ATIS portions of the Penn Treebank.

  Note the formatting differences for the part-of-speech tags; such small differences are common and must be dealt with in processing treebanks. The use of LISP-style parenthesized notation for trees is extremely common and resembles the bracketed notation. For those who are not familiar with it we show a standard node-and-line tree representation in Fig. 5.2.5.

```
((S
  (NP-SBJ (DT That)
    (JJ cold) (, ,)
    (JJ empty) (NN sky) )
  (VP (VBD was)
    (ADJP-PRD (JJ full)
      (PP (IN of)
        (NP (NN fire)
          (CC and)
          (NN light) ))))
  (. .) ))
                    (a)

((S
  (NP-SBJ The/DT flight/NN )
  (VP should/MD
    (VP arrive/VB
      (PP-TMP at/IN
        (NP eleven/CD a.m/RB ))
      (NP-TMP tomorrow/NN )))))
                    (b)
```

Fig. 5.2.4 : Parsed sentences from the LDC Treebank3 version of the Brown (a) and ATIS (b) corpora.

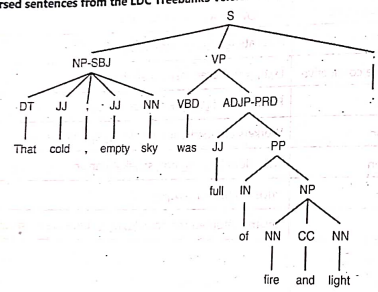


Fig. 5.2.5 : The tree corresponding to the Brown corpus sentence in the previous figure.

- Fig. 5.2.6 shows a tree from the Wall Street Journal. This tree shows another feature of the Penn Treebanks : the use of traces (-NONE- nodes) to marklong-distance dependencies or syntactic movement.
- For example, quotations often follow a quotative verb like say. But in this example, the quotation "We would have to wait until we have collected on those assets" precedes the words he said.
- An empty S containing only the node -NONE- marks the position after said where the quotation sentence often occurs. This empty node is marked (in Treebanks II and III) with the index 2, as is the quotation S at the beginning of the sentence.
- Such co-indexing may make it easier for some parsers to recover the fact that this fronted or topicalized quotation is the complement of the verb said. A similar -NONE- node marks the fact that there is no syntactic subject right before the verb to wait; instead, the subject is the earlier NP We. Again, they are both co-indexed with the index 1.
- The Penn Treebank II and Treebank III releases added further information to make it easier to recover the relationships between predicates and arguments. Certain phrases were marked with tags indicating the grammatical function of the phrase(as surface subject, logical topic, cleft, non-VP predicates) its presence in particular text categories (headlines, titles), and its semantic function (temporal phrases, locations) (Marcus et al. 1994; Bies et al. 1995).

- Fig. 5.2.6 shows examples of the-SBJ (surface subject) and -TMP (temporal phrase) tags. Fig. 5.2.5 shows in addition the -PRD tag, which is used for predicates that are not VPs (the one in Fig. 5.2.5 is an ADJP).

```
( (S (`` ``)
    (S-TPC-2
      (NP-SBJ-1 (PRP We) )
      (VP (MD would)
        (VP (VB have)
          (S
            (NP-SBJ (-NONE- *-1) )
            (VP (TO to)
              (VP (VB wait)
                (SBAR-TMP (IN until)
                  (S
                    (NP-SBJ (PRP we) )
                    (VP (VBP have)
                      (VP (VBN collected)
                        (PP-CLR (IN on)
                          (NP (DT those)(NNS assets)))))))))))))
    (, ,) ('' '')
    (NP-SBJ (PRP he) )
    (VP (VBD said)
      (S (-NONE- *T*-2) ))
    (. .) ))
```

Fig. 5.2.6 : A sentence from the Wall Street Journal portion of the LDC Penn Treebank. Note the use of the empty -NONE- nodes

5.2.7 Universal Dependency Treebanks

- As with constituent-based methods, treebanks play a critical role in the development and evaluation of dependency parsers. Dependency treebanks have been created using similar approaches to having human an notators directly generate dependency structures for a given corpus, or using automatic parsers to provide an initial parse and then having annotators hand correct those parsers. We can also use a deterministic process to translate existing constituent based treebanks into dependency trees through the use of head rules.
- For the most part, directly annotated dependency treebanks have been created for morphologically rich languages such as Czech, Hindi and Finnish that lend themselves to dependency grammar approaches, with the Prague Dependency Treebank for Czech being the most well-known effort.
- The major English dependency treebanks have largely been extracted from existing resources such as the Wall Street Journal sections of the Penn Treebank. The more recent Onto Notes project extends this approach going beyond traditional news text to include conversational telephone speech, weblogs, use net newsgroups, broadcasts, and talk shows in English, Chinese and Arabic.

- The translation process from constituent to dependency structures has two subtasks : identifying all the head-dependent relations in the structure and identifying the correct dependency relations for these relations. The first task relies heavily on the use of head rules first developed for use in lexicalized probabilistic parsers. Here's a simple and effective algorithm from.
  1. Mark the head child of each node in a phrase structure, using the appropriate head rules.
  2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.
- When a phrase-structure parse contains additional information in the form of grammatical relations and function tags, as in the case of the Penn Treebank, these tags can be used to label the edges in the resulting tree. When applied to the parse tree in Fig. 5.2.6 this algorithm would produce the dependency structure in Fig. 5.2.7.

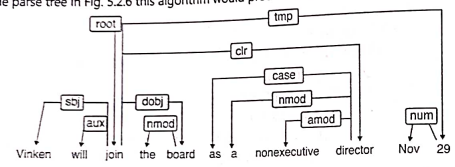


Fig 5.2.7 : Example : Dependency Structure

- The primary shortcoming of these extraction methods is that they are limited by the information present in the original constituent trees. Among the most important issues are the failure to integrate morphological information with the phrase structure trees, the inability to easily represent non-projective structures, and the lack of internal structure to most noun-phrases, as reflected in the generally flat rules used in most treebank grammars. For these reasons, outside of English, most dependency treebanks are developed directly using human annotators.

5.3 Word Sense Disambiguation

- Generating sense labeled corpora like SemCor is quite difficult and expensive. An Alternative class of WSD algorithms, knowledge-based algorithms, rely solely on WordNet or other such resources and don't require labeled data.
- While supervised algorithms generally work better, knowledge-based methods can be used in languages or domains where thesauruses or dictionaries but not sense labeled corpora are available.
- Word sense disambiguation is a very important task in natural language processing applications such as machine translation and information retrieval.
- In this paper, different approaches to this problem are described and summarized, and another method for Turkish using WordNet is proposed. In this method, Lesk algorithm is used but it is extended in a way that it exploits the hierarchical structure of WordNet.

5.3.1 Lesk Algorithm

- The Lesk algorithm is the oldest and most powerful knowledge-based WSD method, and is a useful baseline. Lesk is really a family of algorithms that choose the sense whose dictionary gloss or definition shares the most words with the target word's neighborhood. Fig. 5.3.1 shows the simplest version of the algorithm, often called the Simplified Lesk algorithm.

- As an example of the Lesk algorithm at work, consider disambiguating the word bank in the following context:

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word
  best-sense ← most frequent sense for word
  max-overlap ← 0
  context ← set of words in sentence
  for each sense in senses of word do
    signature ← set of words in the gloss and examples of sense
    overlap ← COMPUTEOVERLAP(signature, context)
    if overlap > max-overlap then
        max-overlap ← overlap
        best-sense ← sense
  end
  return(best-sense)
```

Fig. 5.3.1 : The Simplified Lesk algorithm. The COMPUTEOVERLAP function returns the number of words in common between two sets, ignoring function words or other words on a stop list. The original Lesk algorithm defines the context in a more complex way.

- One of the algorithms to solve word sense disambiguation is the Lesk algorithm. It has a huge corpus in the background (generally WordNet is used) that contains definitions of all the possible synonyms of all the possible words in a language. Then it takes a word and the context as input and finds a match between the context and all the definitions of the word. The meaning with the highest number of matches with the context of the word will be returned.
- For example, suppose we have a sentence such as "We play only soccer" in a given text. Now, we need to find the meaning of the word "play" in this sentence. In the Lesk algorithm, each word with ambiguous meaning is saved in background synsets. In this case, the word "play" will be saved with all possible definitions. Let's say we have two definitions of the word "play":
  1. **Play :** Participating in a sport or game
  2. **Play :** Using a musical instrument
- Then, we will find the similarity between the context of the word "play" in the text and both of the preceding definitions using text similarity techniques. The definition best suited to the context of "play" in the sentence will be considered the meaning or definition of the word. In this case, we will find that our first definition fits best in context, as the words "sport" and "game" are present in the preceding sentences.
- **Example :** The bank can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

  given the following two WordNet senses :

$Bank^1$	Gloss :	A financial institution that accepts deposits and channels the money into leading activities
	Examples :	"he cashed a check at the bank", "that bank holds the mortage on my home"
$Bank^2$	Gloss :	Sloping land (especially the slope beside a body of water)
	Examples :	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

- Sense bank[1] has two non-stop words overlapping with the context in above example : deposits and mortgage, while sense bank has zero words, so sense bank is chosen.
- There are many obvious extensions to Simplified Lesk, such as weighing the overlapping words by IDF (inverse document frequency) to down weight frequent words like function words; best performing is to use word embedding cosine instead of word overlap to compute the similarity between the definition and the context. Modern neural extensions of Lesk use the definitions to compute sense embeddings that can be directly used instead of SemCor-training embeddings.

5.3.2 Walker's Algorithm

It is a Thesaurus Based approach.

Step 1 : For each sense of the target word find the thesaurus category to which that sense belongs.

Step 2 : Calculate the score for each sense by using the context words. A context words will add 1 to the score of the sense if the thesaurus category of the word matches that of the sense.

- E.g. The money in this **bank** fetches an interest of 8% per annum
- Target word : **bank**
- Clue words from the context: **money, interest, annum, fetch**

	Sense1 : Finance	Sense2: Location
Money	+1	0
Interest	+1	0
Fetch	0	0
Annum	+1	0
Total	3	0

Context words add 1 to the sense when the topic of the word matches that of the sense

5.3.3 WordNets for Word Sense Disambiguation

- The concept of sense ambiguity means that a word which has more than one meaning is used in a context and it needs to be clarified that which sense is actually referred. Word sense disambiguation (WSD) is the concept of identifying which sense of a word is used in a sentence or context.
- Word sense disambiguation is a very important task in natural language processing applications such as machine translation and information retrieval.
- WordNet is a lexical database for the English language, which was created by Princeton, and is part of the NLTK corpus. It is a machine-readable database of words which can be accessed from most popular programming languages (C, C#, Java, Ruby, Python etc.). WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings.

- WordNet is not like your traditional dictionary. WordNet focuses on the relationship between words along with their definitions, and this makes a WordNet a network instead of a list. NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets.
- In the WordNet network, the words are connected by linguistic relations. These linguistic relations (hypernym, hyponym, meronym, holonym and other fancy sounding stuff), are WordNet's secret sauce. They give you powerful capabilities that are missing in an ordinary dictionary/thesaurus.

Homonymy

- It is a relation between words that have the same form (and the same PoS) but unrelated meanings

  e.g. *bank* (the financial institution, the river bank)
- It causes ambiguities for the interpretation of a sentence since it defines a set of different lexemes with the same orthographic form (bank1, bank2,..)
- Related properties are homophony (same pronunciation but different orthography, e.g. *be-bee*) and homography (same orthography but different pronunciation pésca/pèsca)

Polysemy

- It happens when a lexeme has more related meanings
  - It depends on the word etymology (unrelated meanings usually have a different origin) - e.g. *bank/data bank/blood bank*
- For polysemous lexemes we need to manage all the meanings
  - We should define a method to determine the meanings (their number and semantics) and if they are really distinct (by experts in lexicography)
  - We need to describe the eventual correlations among the meanings
  - We need to define how the meanings can be distinguished in order to attach the correct meaning to a word in a given context (word sense disambiguation)

Synonymy

- It is a relationship between two distinct lexemes with the same meaning (i.e. they can be substituted for one another in a given context without changing its meaning and correctness)

  e.g. I received a gift/present
- The substitutability may not be valid for any context due to small semantic differences (e.g. *price/fare of a service – the bus fare/the ticket price*)
- In general substitutability depends on the "semantic intersection" of the senses of the two lexemes and, in some cases, also by social factors (*father/dad*)

Hyponymy

- It is a relationship between two lexemes (more precisely two senses) such that one denotes a subclass of the other
  - car, vehicle – shark, fish – apple, fruit
  - The relationship is not symmetric
    a) The more specialized concept is the hyponym of the more general one

b) The more general concept is the hypernym of the more specialized one

- Hyponymy (hypernymy) is the basis for the definition of a taxonomy (a tree structure that defines inclusion relationships in an object ontology) even if it is not properly a taxonomy

a) The definition of a formal taxonomy would require a more uniform/rigorous formalism in the interpretation of the inclusion relationship

b) However the relationship defines a inheritance mechanism of the properties from the ancestors of a given a concept in the hierarchy

Review Questions

Q.1 Explain natural language toolkit (NLTK) with is libraries.

Q.2 What is significance use of TextBlob library, with example.

Q.3 Which types of tasks are performed by Gensim library? Give one example.

Q.4 What are the different lexical knowledge networks? Explain Indo-WordNet and VerbNet with examples.

Q.5 Explain Lesk Algorithm with example.

□□□

6

Unit VI

Applications of NLP

Syllabus

Machine Translation : Rule based techniques, Statistical Machine Translation (SMT), Cross Lingual Translation Sentiment Analysis, Question Answering, Text Entailment, Discourse Processing, Dialog and Conversational Agents, Natural Language Generation

6.1 Machine Translation

- Machine translation (MT) is one of the "holy grail" problems in artificial intelligence, with the potential to transform society by facilitating communication between people anywhere in the world. As a result, MT has received significant attention and funding since the early 1950s. However, it has proved remarkably challenging, and while there has been substantial progress towards usable MT systems especially for high-resource language pairs like English-French we are still far from translation systems that match the nuance and depth of human translations.
- Machine translation approaches can be broadly classified into the following four categories (Fig. 6.1.1) :

1. Direct machine translation
2. Rule-based translation
3. Corpus-based translation
4. Knowledge-based translation

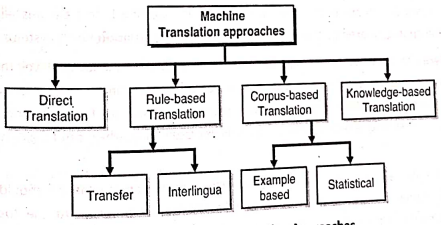


Fig. 6.1.1 : Machine Translation Approaches

Machine translation can be formulated as an optimization problem :

$$\hat{w}^{(t)} = \underset{w^{(t)}}{\text{argmax}}\ \Psi\left(w^{(s)}, w^{(t)}\right)$$

where $w^{(s)}$ is a sentence in a source language, $w^{(t)}$ is a sentence in the target language, and Ψ is a scoring function. As usual, this formalism requires two components: a decoding algorithm for computing $w^{(t)}$, and a learning algorithm for estimating the parameters of the scoring function Ψ.

- Decoding is difficult for machine translation because of the huge space of possible translations. We have faced large label spaces before: for example, in sequence labeling, the set of possible label sequences is exponential in the length of the input.
- In these cases, it was possible to search the space quickly by introducing locality assumptions: for example, that each tag depends only on its predecessor, or that each production depends only on its parent. In machine translation, no such locality assumptions seem possible: human translators reword, reorder, and rearrange words; they replace single words with multi-word phrases, and vice versa.
- This flexibility means that in even relatively simple translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing with this complexity are described in section 6.2.

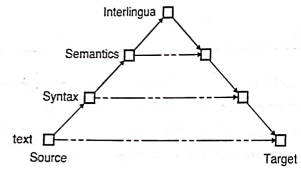


Fig. 6.1.2 : The Vauquois Pyramid

- Estimating translation models is difficult as well. Labeled translation data usually comes in the form parallel sentences, e.g.,

$$w^{(s)} = \text{A vinay le gusta las manzanas}$$

$$w^{(t)} = \text{vinaly likes apple}$$

- A useful feature function would note the translation pairs (gusta, likes), (manzanas, apples), and even (Vinay, Vinay). But this word-to-word **alignment** is not given in the data. One solution is to treat this alignment as a **latent variable**; this is the approach taken by classical **statistical machine translation** (SMT) systems.
- The **Vauquois Pyramid** is a theory of how translation should be done. At the lowest level, the translation system operates on individual words, but the horizontal distance at this level is large, because languages express ideas differently. If we can move up the triangle to syntactic structure, the distance for translation is reduced; we then need only produce target-language text from the syntactic representation, which can be as simple as reading off a tree.
- Further up the triangle lies semantics; translating between semantic representations should be easier still, but mapping between semantics and surface text is a difficult, unsolved problem. At the top of the triangle is ***interlingua***, a semantic representation that is so generic that it is identical across all human languages. Philosophers debate whether such a thing as interlingua is really possible (e.g., Derrida, 1985).
- While the first-order logic representations might be thought to be language independent, they are built on an inventory of predicates that are suspiciously similar to English words (Nirenburg and Wilks, 2001). Nonetheless, the idea of linking translation and semantic understanding may still be a promising path, if the resulting translations better preserve the meaning of the original text.

There are two main criteria for a translation, summarized in Table 6.1.1.

Table 6.1.1 : Adequacy and fluency for translations of the Spanish sentence *A Vinay le gusta Python.*

	Adequate?	Fluent?
To vinay it like python	Yes	No
Vinay debugs memory leaks	No	Yes
Vinaly likes pyton	Yes	Yes

- **Adequacy :** The translation $w^{(t)}$ should adequately reflect the linguistic content of $w^{(s)}$. For example, if $w^{(s)}$ = A Vinay le gusta Python, the reference translation is w(t) = Vinay likes Python. However, the gloss, or word-for-word translation $w^{(t)}$ = To Vinay it like Python is also considered adequate because it contains all the relevant content. The output $w^{(t)}$ = Vinay debugs memory leaks is not adequate.
- **Fluency :** The translation $w^{(t)}$ should read like fluent text in the target language. By this criterion, the gloss $w^{(t)}$ = To Vinay it like Python will score poorly, and $w^{(t)}$ = Vinay debugs memory leaks will be preferred.

6.1.1 Rule Based Techniques

- **Rule-based machine translation** or **rules-based machine translation** (RBMT) is a machine translation approach based on hardcoded linguistic rules.
- Rule based Machine Translation (RBMT) uses linguistic information about source and target languages. It is used most commonly for the creation of dictionaries and grammar programs. RBMT uses linguist rules to break down the content and produces more predictable output for terminology. Rule based engines don't require a bilingual corpus to create the translation system.
- Rule-based MT systems parse the source text and produce an intermediate representation, which may be a parse tree or some abstract representation. The target language text is generated from the intermediate representation. These systems rely on specification of rules for morphology, syntax, lexical selection and transfer, semantic analysis and generation, and are hence called rule-based systems.
- These systems make use of extensive lexicons equipped with morphological, syntactic and semantic information, and a large set of rules to map the input text to intermediate representation. Examples of rule-based include the Ariane and SUSY systems. Depending on the intermediate representation used, these systems are further categorized as follows :

1. Transfer-based machine translation
2. Interlingua machine translation

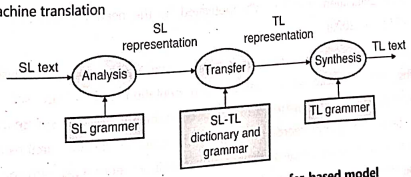


Fig. 6.1.3 : Schematic diagram of the transfer-based model

Transfer-based Translation :

- A direct translation model maps directly between source and target language without producing an intermediate representation. An alternative strategy is to use transfer- based translation models. These models transform the structure of the input to produce a representation that matches the rules of the target language.
- This transformation requires an understanding of the differences between the source and target language. In order to get the structure of the input, some form of parse is needed. Therefore, transfer-based translation involves parsing of the source text.
- The source language parse structure is then transferred into the target language structure. The generation of the target language text is facilitated by a generation module. Thus, a transfer-based machine translation system has the following three components :
  1. Analysis to produce source language structure
  2. Transfer-To transfer the source language representation to a target level representation
  3. Generation-To generate target language text using target level structure
- The first stage analyses the source text and produces a structure confirming the rules of source language. Details of the analysis vary from system to system. It may involve morphological, syntactic, and semantic analyses. As this stage involves parsing of source text, syntactic ambiguities and lexical ambiguities are better resolved in this approach than in the direct translation approach. The second stage transfers source language representation into target language representation. All the language-pair specific peculiarities are handled by the transfer component. The third stage is responsible for generating the actual target language text.
- The main advantage of this approach is its modular structure. The analysis of source language text (Le., the parser) is independent of target language generator. All language-pair specific differences are captured in the transfer stage. In order to provide translation capability among a set of languages, we need an analyzer and a generator component for each language and a transfer component for each pair of such languages. For example, to provide translation capability for six languages, we need six analysers, six generators, and 30 transfer components, as opposed to 30 complete transfer systems needed in a direct translation approach.
- A second advantage is that transfer systems can easily handle ambiguities that carry over from one language to another. For example, we need not manually resolve the PP-attachment ambiguity in the sentence, *The girl plucked a flower with stick*, while translating it into French. Likewise, the lexical ambiguity in the Hindi sentence, *Mujhe sona achchha lagta hai*, need not be resolved while translating it into Urdu.
- Transfer systems also help in resolving certain lexical ambiguities. Consider sentence (*Book a ticket for me*). The parse tree generated by the transfer system makes it clear that 'book' is used as verb in this sentence and not as a noun. This information prevents book' from being translated into 'kitaab in Hindi.
- A further advancement in the transfer-based approach is the use of reversible representations and processing rules. Feature-structure unification, was basically motivated by the need for reversible transformation in MT systems (Jurafsky and Martin 2000).
- Use of unification and constraint-based grammar formalisms simplify the rules of analysis, transformation, and generation (Hutchins 1995). Instead of a multi-level representation and a large set of rules (for mapping morphological representation to syntactic tree, syntactic tree to semantic tree, and source language representation into an equivalent target language representation), which apply in specific circumstances and to specific representations, there are monostratal representations and an abstract set of rules, with conditions and constraints incorporated into specific lexical entries.

- The reversible nature of grammar components has the advantage that for a given language the same grammar can be used both for analysis as well as for generation. The essential idea of the structural transfer algorithm is that SOV in English becomes SVO in Hindi, and post modifiers in English become pre modifiers in Hindi (Rao et al. 1998). Fig. 6.1.4 shows an example of the structural transfer for sentence (Khushbu slept in the garden).

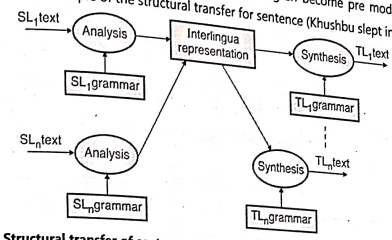


Fig. 6.1.4 : Structural transfer of sentence (Khushbu slept in the garden) from Hindi to English

6.1.2 Statistical Machine Translation (SMT)

- Statistical machine translation (SMT) generates translations using statistical methods and models based on bilingual text corpora, such as the Canadian Hansard corpus or the record of the European Parliament. Statistical engines don't analyze text based on language rules. Rather, they are built by analyzing bilingual corpus. This method requires a large volume of bilingual content.
- The previous section introduced adequacy and fluency as the two main criteria for machine translation. A natural modeling approach is to represent them with separate scores,

$$\Psi\left(w^{(s)}, w^{(t)}\right) = \Psi_A\left(w^{(s)}, w^{(t)}\right) + \Psi_F\left(w^{(t)}\right) \quad \ldots (6.1.1)$$

- The fluency score Ψ_F need not even consider the source sentence; it only judges $w^{(t)}$ on whether it is fluent in the target language. This decomposition is advantageous because it makes it possible to estimate the two scoring functions on separate data.
- While the adequacy model must be estimated from aligned sentences - which are relatively expensive and rare - the fluency model can be estimated from monolingual text in the target language. Large monolingual corpora are now available in many languages, thanks to resources such as Wikipedia.
- An elegant justification of the decomposition in Equation [01] is provided by the **noisy channel model,** in which each scoring function is a log probability:

$$\Psi_A\left(w^{(s)}, w^{(t)}\right) \triangleq \log p_{s|T}\left(w^{(s)}, w^{(t)}\right) \quad \ldots (2)$$

$$\Psi_F\left(w^{(t)}\right) \triangleq \log p_T\left(w^{(t)}\right) \quad \ldots (3)$$

$$\Psi\left(w^{(s)}, w^{(t)}\right) = \log p_{S|T}\left(w^{(s)}, w^{(t)}\right) + \log p_T\left(w^{(t)}\right) = \log p_{S,T}\left(w^{(s)}, w^{(t)}\right) \quad \ldots (4)$$

- By setting the scoring functions equal to the logarithms of the prior and likelihood, their sum is equal to log $p_{S,T}$, which is the logarithm of the joint probability of the source and target. The sentence $w^{(t)}$ that maximizes this joint probability is also the maximizer of the conditional probability $p_{T|S}$, making it the most likely target language sentence, conditioned on the source.

- The noisy channel model can be justified by a generative story. The target text is originally generated from a probability model P_T. It is then encoded in a "noisy channel" $P_{S|T}$, which converts it to a string in the source language. In decoding, we apply Bayes' rule to recover the string $w^{(t)}$ that is maximally likely under the conditional probability $P_{S|T}$. Under this interpretation, the target probability P_T is just a language model. The only remaining learning problem is to estimate the translation model $P_{S|T}$.

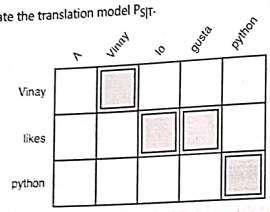


Fig. 6.1.5 : An example word-to-word alignment

6.1.2(A) Statistical Translation Modelling

- The simplest decomposition of the translation model is word-to-word: each word in the source should be aligned to a word in the translation. This approach presupposes an **alignment** $A\left(w^{(s)}, w^{(t)}\right)$, which contains a list of pairs of source and target tokens. For example, given $w^{(s)}$ = *A Vinay le gusta Python* and $w^{(t)}$ = *Vinay likes Python*, one possible word-to-word alignment is,

$$A\left(w^{(s)}, w^{(t)}\right) = \{(A, \phi), (Vinay, vinay), (le, likes), (gusta, likes), (Python, Python)\}$$

- This alignment is shown in Fig. 6.1.5 Another, less promising, alignment is :

$$A\left(w^{(s)}, w^{(t)}\right) = \{(A, Vinay), (Vinay, likes), (le, python), (gusta, \phi), (Python, \phi)\}$$

- Each alignment contains exactly one tuple for each word in the source, which serves to explain how the source word could be translated from the target, as required by the translation probability $P_{S|T}$. If no appropriate word in the target can be identified for a source word, it is aligned to ϕ - as is the case for the Spanish function word a in the example, which glosses to the English word to. Words in the target can align with multiple words in the source, so that the target word likes can align to both *le* and *gusta* in the source.
- The joint probability of the alignment and the translation can be defined conveniently as,

$$p\left(w^{(s)}, w^{(t)}\right) = \prod_{m=1}^{M^{(s)}} p\left(w_m^{(s)}, a_m \mid w_{a_m}^{(s)}, m, M^{(s)}, M^{(t)}\right)$$

$$= \prod_{m=1}^{M^{(s)}} p\left(a_m \mid m, M^{(s)}, M^{(t)}\right) \times p\left(w_m^{(s)} \mid w_{a_m}^{(t)}\right)$$

This probability model makes two key assumptions :

- The alignment probability factors across tokens,

$$p\left(A \mid w^{(s)}, w^{(t)}\right) = \prod_{m=1}^{M^{(s)}} p\left(a_m \mid m, M^{(s)}, M^{(t)}\right)$$

This means that each alignment decision is independent of the others, and depends only on the index m, and the sentence lengths $M^{(s)}$ and $M^{(t)}$.

- The translation probability also factors across tokens,

$$p\left(w^{(s)} \mid w^{(s)}, A\right) = \prod_{m=1}^{M^{(s)}} \left(w_m^{(s)} \mid w_{a_m}^{(t)}\right).$$

- so that each word in $w^{(s)}$ depends only on its aligned word in $w^{(t)}$. This means that translation is word-to-word, ignoring context. The hope is that the target language model $p(w^{(t)})$ will correct any disfluencies that arise from word-to-word translation.
- To translate with such a model, we could sum or max over all possible alignments,

$$p\left(w^{(s)}, w^{(t)}\right) = \sum_A p\left(w^{(s)}, w^{(t)}, A\right) = p\left(w^{(t)}\right) \sum_A p(A) \times p\left(w^{(s)} \mid w^{(t)}, A\right)$$

$$\geq p\left(w^{(t)}\right) \max_A p(A) \times p\left(w^{(s)} \mid w^{(t)}, A\right)$$

- The term p(A) defines the prior probability over alignments. A series of alignment models with increasingly relaxed independence assumptions was developed by researchers at IBM in the 1980s and 1990s, known as IBM Models 1-6 (Och and Ney, 2003). IBM Model 1 makes the strongest independence assumption:

$$P\left(a_m \mid m, M^{(s)}, M^{(t)}\right) = \frac{1}{M^{(t)}}$$

- In this model, every alignment is equally likely. This is almost surely wrong, but it results in a convex learning objective, yielding a good initialization for the more complex alignment models (Brown et al., 1993; Koehn, 2009).

6.2 Cross Lingual Machine Translation

Cross-lingual learning is a paradigm for transferring knowledge from one natural language to another. The transfer of knowledge can help us overcome the lack of data in the target languages and create intelligent systems and machine learning models for languages, where it was not possible previously.

Cross-lingual Language Model (XLM)

- The model uses the **same shared vocabulary** for all the languages. This helps in establishing a common embedding space for tokens from all languages. Hence, it is evident that languages that have the same script (alphabets), or similar words map better to this common embedding space.
- For tokenizing the corpora, Byte-Pair Encoding (BPE) subword algorithm is used.
- Besides subword, XLM also feed position embeddings (represent the position of a sentence) and language embeddings (represent different language) into different Language Model (LM) to learn text representation. Those LM are :
  - Causal Language Modeling (CLM)
  - Masked Language Modeling (MLM)

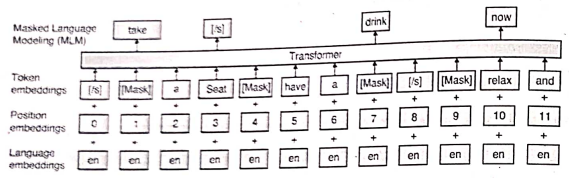
- Translation Language Modeling (TLM)

1. Causal Language Modeling (CLM)

This is the regular Language Modeling objective where we maximize the probability of a token ***x_t*** to appear at the '***t***th' position in a given sequence given all the tokens ***x_ < t*** (all the tokens preceding the '***t***th' token) in that sequence, i.e.

$$\max_{\theta} \log p\theta(x) = \sum_{t=1}^{T} \log p\theta(x_t \mid x < t)$$

2. Masked Language Modeling (MLM) :

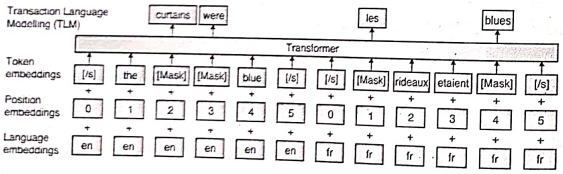


This is a type of the Denoising Autoencoding objective, also known as the Cloze task. Here, we maximize the probability of a given masked token ***x_t*** to appear at the '***t***th' position in a given sequence given all the tokens in that sequence, ***x_hat***. i.e.

$$\max_{\theta} \log p\theta(\bar{x} \mid \hat{x}) = \sum_{t=1}^{T} m_t \log p\theta(x_t \mid \hat{x})$$

BERT and RoBERTa are trained on this objective.

3. Translation Language Modeling (TLM)



The CLM and MLM tasks work well on monolingual corpora, however, they do not take advantage of the available parallel translation data. Hence, the authors propose a Translation Language Modeling objective wherein we take a sequence of parallel sentences from the translation data and **randomly mask tokens from the source as well as from the target sentence**. For example, in the figure above, we have masked words from English as well as from the French sentence. **All the words in the sequence contribute to the prediction of a given masked word**, hence establishing a cross-lingual mapping among the tokens.

6.3 Sentiments Analysis

- We will now discuss several concepts, techniques, and examples with regard to our second major topic in this chapter, sentiment analysis. Textual data, even though unstructured, mainly has two broad types of data points: factual based (objective) and opinion based (subjective).
- We briefly talked about these two categories at the beginning of this chapter when I introduced the concept of sentiment analysis and how it works best on text that has a subjective context. In general, social media, surveys, and feedback data all are heavily opinionated and express the beliefs, judgement, emotion, and feelings of human beings.
- Sentiment analysis, also popularly known as opinion analysis/mining , is defined as the process of using techniques like NLP, lexical resources, linguistics, and machine learning (ML) to extract subjective and opinion related information like emotions, attitude, mood, modality, and so on and try to use these to compute the polarity expressed by a text document.
- By polarity , I mean to find out whether the document expresses a positive, negative, or a neutral sentiment. More advanced analysis involves trying to find out more complex emotions like sadness, happiness, anger, and sarcasm.
- Typically, sentiment analysis for text data can be computed on several levels, including on an individual sentence level, paragraph level, or the entire document as a whole. Often sentiment is computed on the document as a whole or some aggregations are done after computing the sentiment for individual sentences.
- Polarity analysis usually involves trying to assign some scores contributing to the positive and negative emotions expressed in the document and then finally assigning a label to the document based on the aggregate score. We will depict two major techniques for sentiment analysis here :
  - Supervised machine learning
  - Unsupervised lexicon-based
- The key idea is to learn the various techniques typically used to tackle sentiment analysis problems so that you can apply them to solve your own problems. We will see how to re-use the concepts of supervised machine learning based classification algorithms here to classify documents to their associated sentiment.
- We will also use lexicons, which are dictionaries or vocabularies specially constructed to be used for sentiment analysis, and compute sentiment without using any supervised techniques.
- We will be carrying out our experiments on a large real-world dataset pertaining to movie reviews, which will make this task more interesting.
- We will compare the performance of the various algorithms and also try to perform some detailed analytics besides just analyzing polarity, which includes analyzing the subjectivity, mood, and modality of the movie reviews. Without further delay, let's get started!

6.4 Question Answering Systems

- Question Answering Systems (QAS) are built upon the principle of Question Answering, based on using techniques from NLP and information retrieval (IR).
- QAS is primarily concerned with building robust and scalable systems that provide answers to questions given by users in natural language form.

- Imagine being in a foreign country, asking a question to your personalized assistant in your phone in pure natural language, and getting a similar response from it.
- This is the ideal state toward which researchers and technologists are working. Some success in this field has been achieved with personalized assistants like Siri and Cortana, but their scope is still limited because they understand only a subset of key clauses and phrases in the entire human natural language.
- To build a successful QAS, you need a huge knowledgebase consisting of data about various domains. Efficient querying systems into this knowledgebase would be leveraged by the QAS to provide answers to questions in natural language form. Creating and maintaining a queryable vast knowledgebase is extremely difficult hence, you find the rise of QAS in niche domains like food, healthcare, e-commerce, and so on. Chatbots are one emerging trend that makes extensive use of QAS.

6.5 Textual Entailment

- Textual entailment recognition is the task of deciding, given two text fragments, whether the meaning of one text is entailed (can be inferred) from another text. This task captures generically a broad range of inferences that are relevant for multiple applications.
- For example, a QA system has to identify texts that entail the expected answer :

Question Expected answer form

Who killed Kennedy ? >> X killed Kennedy

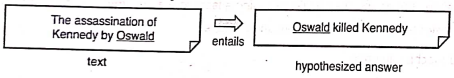


Fig. 6.5.1

- In IE entailment holds between different text variants that express the same target relation: (X kill Y X acquire Y)
- In MDS a redundant sentence or expression, to be omitted from the summary, should be entailed from other expressions in the summary.
- In IR the concept denoted by a query expression should be entailed from relevant retrieved documents.
- In MT evaluation a correct translation should be semantically equivalent to the gold standard translation, and thus both translations have to entail each other.
- Thus, in a similar spirit to WSD and NER which are recognized as generic tasks, modeling textual entailment may consolidate and promote broad research on applied semantic inference.

6.5.1 Textual Entailment Applications

- Question-Answering
- Information Extraction
- Information Retrieval
- Multi-Document Summarization
- Named Entity Recognition

- Temporal and Spatial Normalization
- Semantic Parsing
- Natural Language Generation

6.6 Discourse Processing

- **Discourse processing,** as a subfield of NLP, is a suite of NLP tasks to uncover linguistic structures from texts at several levels, which can support many downstream automated text mining applications.
- Discourse takes various modalities, structures, and mediums. Among the commonly experienced mediums are face-to-face chats, telephone conversations, television news broadcasts, radio news, talk shows, lectures, books, and scientific articles of which the latter form is of particular interest at TIB handled within its next-generation scholarly communication digitalization efforts via the Open Research Knowledge Graph.
- Intriguingly, each of these forms of discourse follow a logical structural nuance depending on the medium. The advancement of the digital age including social media communication has further led to expansion of logical discourse structures.
- These include blog-posts, emails, websites, review sites of products, hotels, restaurants or movies, and, finally, social media streams such as Twitter, Facebook, Reddit, #slack channels, Q&A portals such as Quora or Stack Overflow, etc., with a growing list as new mediums of communication over the World Wide Web are invented.
- Discourse is categorized as either written or speech. It is also categorized as either monologue or conversational which include dialogues. In this context, Discourse Processing then involves identifying the topic structure, the coherence structure, the coreference structure, and the conversation structure the latter specific to conversational discourse.
- Taken together, these structures form the nuts and bolts of specific NLP applications such as text summarization, essay scoring, sentiment analysis, machine translation, information extraction, and question answering.
- In the context of Discourse Processing, Linguists have proposed various theories of Discourse structures which are indeed realized as the annotation framework of natural language toward their machine interpretability for NLP tasks.
- The prominent ones are Segmented Discourse Representation Theory, Discourse Lexicalized Tree Adjoining Grammar which was the model adopted to annotate discourse relations in the Penn Discourse Treebank Corpus comprising over 1 million news articles from the Wall Street Journal, and, last but not the least, Rhetorical Structure Theory .

6.6.1 Dialog and Conversational Agents

Dialogue :

- Dialogue is a conversation between two speakers, also multiparty dialogue is a conversation among more than two speakers.
- Each dialogue consists of a sequence of turns (an utterance by one of the two speakers) Turn-taking requires the ability to detect when the other speaker has finished.

Dialogue Acts :

- Utterances correspond to actions by the speaker, e.g.
  - Constative (answer, claim, confirm, deny, disagree, state)
  - Speaker commits to something being the case
  - Directive (advise, ask, forbid, invite, order, request)
  - Speaker attempts to get listener to do something
  - Commissive (promise, plan, bet, oppose)
  - Speaker commits to a future course of action
  - Acknowledgment (apologize, greet, thank, accept apology)
  - Expresses attitude wrt. some social action
- In practice, much more fine-grained labels are often used, for example: Yes-No Questions, Wh-Questions, Rhetorical Questions, Greetings, Thanks, Yes-Answers, No-Answers, Agreements, Disagreements, Statements, Opinions, Hedges.
- Dialogues have (hierarchical) structure :

  "Adjacency pairs": Some acts (first pair part) typically followed by (set up expectation for) another (second pair part) :

  Question → Answer, Proposal → Acceptance/Rejection, etc.

Conversational Agent :

- A conversational agent is any dialogue system that not only conducts natural language processing but also responds automatically using human language. These agents represent the practical implementation of computational linguistics, usually employed as chatbots over the internet or as portable device assistants. This interpretation/response interaction doesn't have to be conducted just with text.
- The dialogue system can also read from (input channel) and respond with (output channel) speech, graphics, virtual gesture or haptic-assisted physical gestures.

6.7 Natural Language Generation

- **Natural Language Generation** (NLG) is the process of constructing natural language outputs from non-linguistic inputs. The goal of this process can be viewed as the inverse of that of **natural language understanding** (NLU) in that NLG maps from meaning to text, while NLU maps from text to meaning. In doing this mapping, generation visits many of the same linguistic issues, but the inverse orientation distinguishes its methods from those of NLU in two important ways.
- First, the nature of the input to the generation process varies widely from one application to the next. Although the linguistic input to NLU systems may vary from one text type to another, all text is governed by relatively common grammatical rules. This is not the case for the input to generation systems.
- Each generation system addresses a different application with a different input specification. One system may be explaining a complex set of numeric tables while another may be documenting the structure of an object oriented software engineering model. As a result, generation systems must extract the information necessary to drive the generation process.

- Second, while both NLU and NLG must be able to represent a range of lexical and grammatical forms required for the application domain, their use of these representations is different. NLU has been characterized as a process of *hypothesis management* in which the linguistic input is sequentially scanned as the system considers alternative interpretations. Its dominant concerns include ambiguity, under-specification, and ill-formed input.
- These concerns are not generally addressed in generation research because they don't arise. The non-linguistic representations input to an NLG system tend to be relatively unambiguous, well-specified, and well-formed.
- In contrast, the dominant concern of NLG is *choice*. Generation systems must make the following choices:
  - **Content selection** : The system must choose the appropriate content to express from a potentially over-specified input, basing its decision on a specific communicative goal. If the goal was to indicate that the environment is set up, and the reader was a systems engineer, then we'd probably express only the last clause.
  - **Lexical selection** : The system must choose the lexical item most appropriate for expressing particular concepts. In example *Congratulations, you have just compiled and run a simple C program which means that your environment is configured properly.*, for instance, it must choose between the word "configured" and other potential forms including "set up".
  - **Sentence structure**
    - a) **Aggregation :** The system must apportion the selected content into phrase, clause, and sentence-sized chunks. Above example combined the actions of compiling and running into a single phrase.
    - b) **Referring expressions :** The system must determine how to refer to the objects being discussed. As we saw, the decision on how to refer to the program in above example was not trivial.
  - **Discourse structure :** NLG systems frequently deal with multi-sentence discourse, which must have a coherent, discernible structure. Example mentioned above included two propositions in which it was clear that one was giving evidence for the other.
- These issues of choice, taken together with the problem of actually putting linear sequences of words on paper, form the core of the field of NLG. Though it is a relatively young field, it has begun to develop a body of work directed at this core.

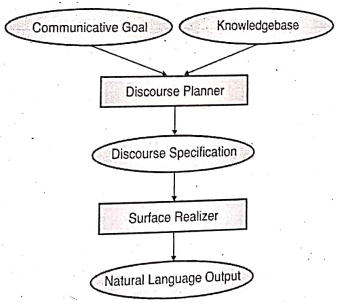


Fig. 6.7.1 : A reference architecture for NLG systems

An architecture for generation :

- The nature of the architecture appropriate for accomplishing the tasks listed in the previous section has occasioned much debate. Practical considerations, however, have frequently led to the architecture shown in **Fig. 6.7.1**. This architecture contains two pipelined components :
  - **Discourse Planner** : This component starts with a communicative goal and makes all the choices discussed in the previous section. It selects the content from the knowledge base and then structures that content appropriately. The resulting discourse plan will specify all the choices made for the entire communication, potentially spanning multiple sentences and including other annotations (including hypertext, figures, etc.).
  - **Surface Realizer :** This component receives the fully specified discourse plan and generates individual sentences as constrained by its lexical and grammatical resources. These resources define the realizer's potential range of output. If the plan specifies multiple-sentence output, the surface realizer is called multiple times.
- This is by no means the only architecture that has been proposed for NLG systems. Other potential mechanisms include AI-style planning and blackboard architectures. Neither is this architecture without its problems. The simple pipeline, for example, doesn't allow decisions made in the planner to be reconsidered during surface realization. Furthermore, the precise boundary between planning and realization is not altogether clear. Nevertheless, we will use it to help organize this chapter. We'll start by discussing the surface realizer, the most developed of the two components, and then proceed to the discourse planner.

Review Question

Q.1 Explain Statistical machine Translation (SMT) with suitable diagram and example.

Q.2 The dominant concern of NLG is *choice*. Which are the choices that Generation systems must make ?

Q.3 How discourse processing uncover linguistic structures from texts at several levels ?

Q.4 Explain Cross Lingual machine translation (XLM) model.

□□□