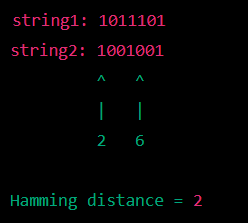
# String similarity metrics

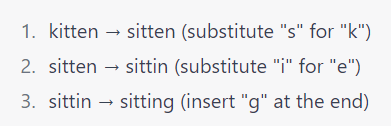
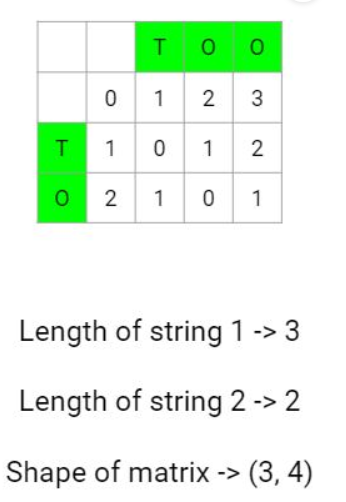
## Hamming distance

* Number of positions at which two strings differ.
* String must have same length.
* Add padding to with shorter if length is different.
* Can be used in spell-checking, plagiarism detection, and DNA sequence analysis



## Levenshtein distance or Minimum edit distance

Number of edit required for that will transform one string to other.

Answer is 3. Answer is 1.

Check if the characters are same

# fill the element with the minimum of (diagonal value, first corresponding value + 1, second corresponding value + 1)

If characters are not same

# fill the element with the minimum of corresponding values + 1

## Jaccard similarity

This measures the size of the intersection of two sets of characters divided by the size of the union of the two sets.

## Cosine similarity

This measures the cosine of the angle between two vectors in a multi-dimensional space, where each string is represented as a vector of character counts.

## Longest common subsequence

This measures the length of the longest sequence of characters that appears in both strings in the same order, but not necessarily consecutively.

Lecture 3:

Manhattan distance better then euclidean distance dealing with outlier

MD = abs difference btn axis

ED = squre root of difference

# Probablistic model:

* N-gram models are a type of Markov mode : probability of a sequence of words based on the probability of the previous n-1 words
* Latent Dirichlet Allocation (LDA) - Generative probabilistic mode, Topic modelling
* Probabilistic context-free grammar (PCFG) : Rule of grammer depends upon the grammer

## Markov Assumption:

the probability of a word depends only on the previous n-1 words

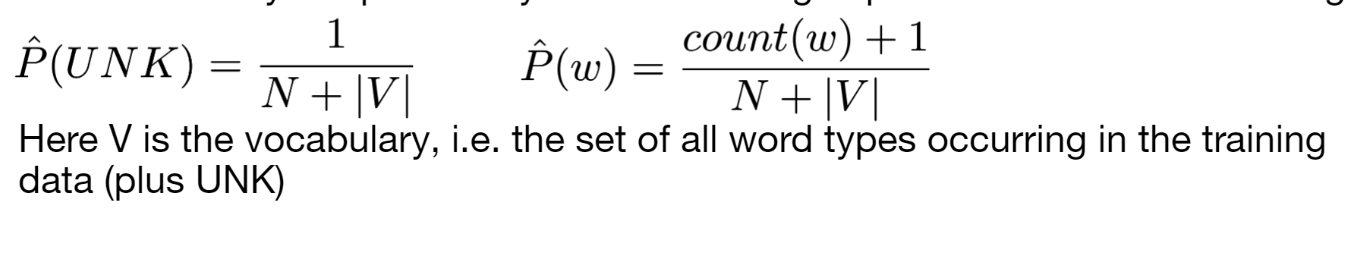
## Problem with unigram model:

* Lack of context:
* Out-of-vocabulary (OOV) words
* Overfitting
* High dimensionality

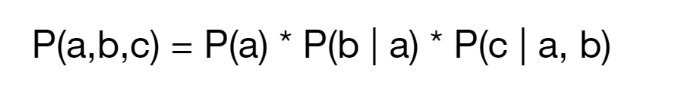
Still, it is very useful in the text classification problem.

## Laplace smoothing

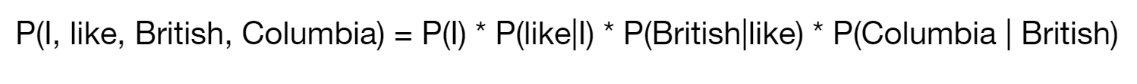
It is known as add-k smoothing. it help to give probablity to unseen data of training set.



Chain of probablity:



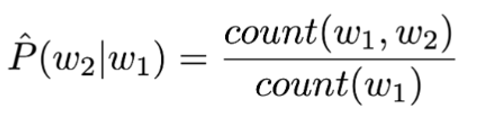
Based on the markov(unigram) :



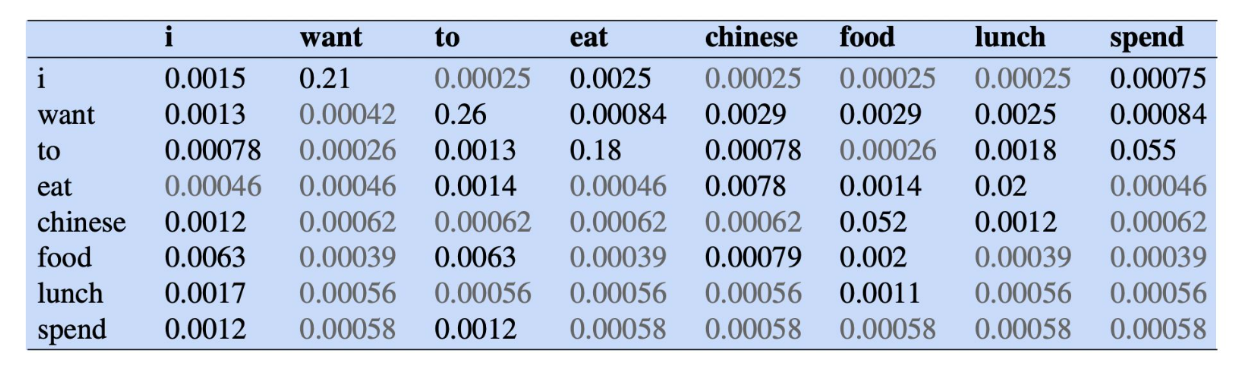
For biggram we need two word after like in 2nd prob.

## Bi gram model

It is also called as Markov chain.

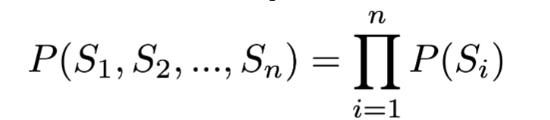


Normalized bi gram model:



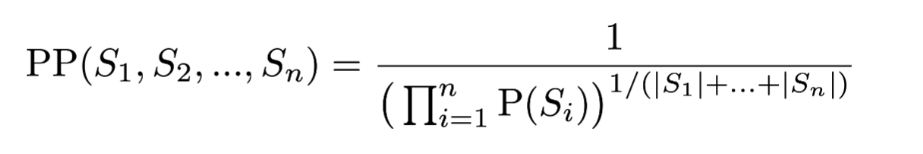
Good way to evaluate the model is test on the unseen data..

Likelihood of a collection of sentences:

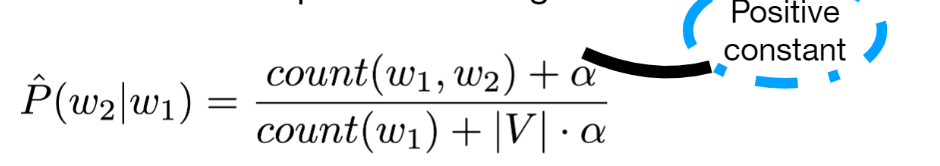


## Perplexity

* For perplexity we measure the normalized inverse above likelihood.
* Perplexity is a measure of how well a probability model is able to predict a given set of observations.
* It is evaluation matrix for the probablity model.



## Adding alpha smoothing



By using hmm with this approach there would be thousands of parameters. So to handle this stituation we can **manage with three state** :

* self.emissions, the list of word types, like “cat”, “dog”
* self.**states**, the list of possible POS tags. “noun”, “verb”
* **Emission distribution** : word is associated with the states like “noun” is probablity distribution for “cat’ and dog may higher then “the”
* 1 x size\_of\_state\_set
* P(w | state = i)
* **Transition probability** : probability of moving from one state to another
* From “noun” to “verb”
* P(state = j | state = i)
* size\_of\_state\_set x size\_of\_state\_set’

With the help of these four HMM to try to find the next word.

## **Generating from an HMM:**

* Generating new observations using new observation using all three states which are similar to used in training.
* It is new training part but we can say prediction part.

### Process steps

The process of generating from an HMM typically involves the following steps:

1. Start in a randomly chosen initial state.
2. Use the transition probabilities to determine the next state.
3. Use the emission probabilities of the current state to determine the next observation.
4. Repeat steps 2 and 3 for a desired number of steps or until a stopping criterion is reached.

Success of right next word : it is based on the **probabilities** learned from the training set.

* This can be achieved by using the **Viterbi Algorithm.**
* P(I,am,Sam; 1,2,3) = P(1|START) \* P(I|1) \* P(2|1) \* P(am|2) \* P(3|2) \* P(Sam|3)

not the same as P(I am Sam)

## **Inference Tasks**

Two states (Rainy and Sunny) and two observations (Umbrella and No Umbrella)

### **Evaluation**

Given a sequence of observations and the HMM parameters, this task calculates the likelihood of the observations being generated by the HMM.

Given observations [Umbrella, No Umbrella, No Umbrella, Umbrella, Umbrella, No Umbrella, No Umbrella].

### **Decoding**

Given a sequence of observations and the HMM parameters, this task finds the most likely sequence of states that generated the observations. This is also known as the Viterbi algorithm.

we have a sequence of observations for the week that is [Umbrella, No Umbrella, No Umbrella, Umbrella, Umbrella, No Umbrella, No Umbrella]. Given the HMM parameters, the most likely sequence of states that generated the observations is [Rainy, Sunny, Sunny, Rainy, Rainy, Sunny, Sunny].

## Hidden and visible states

### Case 1: Inferring Hidden States in an Unsupervised Manner

The states are completely unknown: In this case, the states are not given and we need to infer them from the data in an **unsupervised** manner. This can be done by using techniques such as clustering or Expectation-Maximization (EM) algorithm to group similar observations together and assign each group to a state.

### Case 2: Known Hidden States

The states aren't actually hidden: In this case, the states can be POS tags that come from an annotated corpus. This means that the states are already known and we don't have to infer them from the data.

It is straightforward to estimate model parameters: With the known states, we can use the annotated corpus to estimate the model parameters such as the initial state probabilities, the emission probabilities, and the transition probabilities.

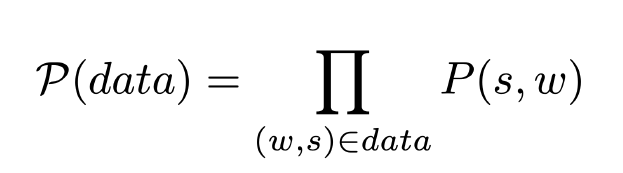
In the 2nd case, it's a **supervised** learning problem, thus it's easier to estimate the model parameters comparing to the 1st case.

## **Training with observable states**

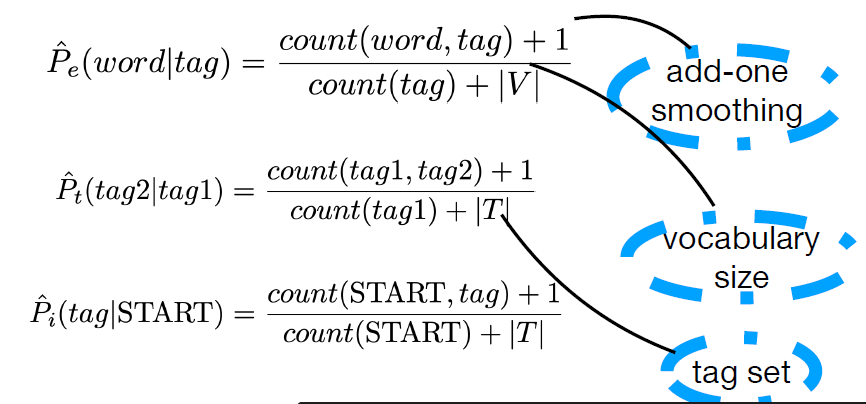
When we train with observable states (e.g. POS tags), the objective is to find model parameters which maximize the joint probability of the states and

emissions in our training corpus:

Where data is a collection of sentences w = w1 ... wk and state sequences s = s1 ... sk



This is called maximum likelihood estimation (MLE)



# SVD

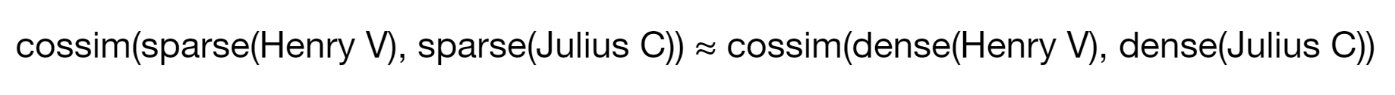
SVD is reducing the dimensionality of the data into 2 D, using linear algebra-based technique. We can say decompose a matrix into its constituent parts.

## Application:

SVD has a wide range of applications in data analysis, including:

* Image compression
* Data mining
* Natural Language Processing (NLP): SVD can be used to analyze text data, such as in topic modeling and sentiment analysis.
* Recommender systems: SVD can be used to factorize a user-item matrix, making it possible to recommend items to users based on their past behavior.

## SVD Aim



## Rank in SVD

* Rank one matrix is product of column and row vector.
* Rank two is sum of two rank one vector.

Rank of a matrix is used to determine the number of singular values and singular vectors needed to decompose the matrix.

A matrix with a higher rank will have more linearly independent rows or columns, require more singular values and singular vectors to decompose.

Turncated SVD:

Approximate a matrix using a smaller number of its singular values and corresponding singular vectors

### Turncated SVD steps

* perform svd on original matrix it will convert into 3 matrix; u, v , s
* Select the number of singular values and singular vectors
* Form the approximation matrix
* Validate the approximation use metrics like Frobenius norm or the relative error to measure the difference between the two matrices.

We can use elbow method to find dimensionality.

# T\_SNE

T-SNE is visulizing the data in 2 or 3 D form and non-linear technique.

* maps the data points to a lower-dimensional space while preserving the local structure of the data

## T\_SNE application

T-SNE has a wide range of applications in data visualization, including:

* Visualizing high-dimensional datasets
* Exploring complex data structures
* Identifying customer segments: T-SNE can be used to visualize patterns in customer data, making it easier to identify customer segments and target marketing efforts.
* Bioinformatics: T-SNE can be used to visualize gene expression data, making it easier to identify patterns and relationships in large, complex datasets.
* Natural Language Processing (NLP): T-SNE can be used to visualize word embeddings, making it easier to understand the relationships between words in a text.

## Hyperparameter

### Perplexity

higher perplexity is better but upto a limit. If value is too high then the it will create one cluster. It should be less then the number of the data points.

### Eplison

Higher eplison put data toghter. Smaller e is better in genreral.

SVD is primarily used for data compression and feature extraction, while T-SNE is primarily used for data visualization

# TF\_IDF:

Importance of word in the documents. We need to remove

## TF

* Number of time a word appear in a documents.
* Like dog appear 10 times in a document.

## IDF

* how common a word appear in the different documents.
* Like dog appear 100 times in 1000 documents so idf will log(100/10000)