Part of Speech Tagging (POS Tagging):

Applications:

* Making assumptions about semantics
* Identifying name entities: Eiffel tower is in Paris.
* Co-reference resolution: Eiffel tower is in Paris. It is 324 m: It refers to Paris.
* Speech Recognition: to check if a sequence of words has a high probability or not.

Markov chains:

* The likelihood of the next word's parts of speech tag in a sentence tends to depend on the parts of speech tag of the previous word.
* Markov chains are a type of stochastic model that describes a sequence of possible events. To get the probability for each event, it needs only the states of the previous events.
* A Markov chain can be depicted as a directed graph.
* The transition probability is the probability of going from one state to another
* A transition matrix is an n by n matrix with n being the number of states in the graph. Each row in the matrix represents transition probabilities of one state to all other states.
* In the transition matrix, all of the transition probabilities in each row should add up to one.
* You can introduce initial states to the transition matrix in the first row, so the matrix becomes n+1 by n.

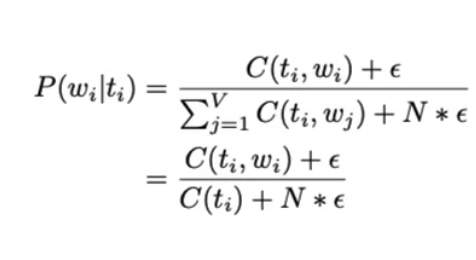
Hidden Markov Models:

* The hidden Markov model also has additional probabilities known as emission probabilities.
* Emission probabilities describe the transition from the hidden states of your hidden Markov model, which are parts of speech, verb, to the observables or the words of your corpus.
* In the emission matrix, each row is designated for one of the hidden states. A column is designated for each of the observables. The emission matrix represents the probabilities for the transition of your n hidden states representing your parts of speech tags to the n words in your corpus. The sum of each row should add up to one.

Populating the transition matrix:

* First count all occurrences of the tag pairs in your training curpus . C(t-1,t)
* Sum all occurrences that start with the first tag. Sum from 1 to j of C(ti-1,tj)
* Calculate the probabilities by dividing those.
* Before computing the probabilities first, you need to prepare your training corpus. Consider each line as a separate sentence. Then add a start token to your lines. Then transform all words to lowercase so the model becomes case sensitive. The punctuations should be intact since there aren’t different tags for different punctuations included.
* Use smoothing to ovoid zero division and having zero probabilities by adding a small value epsilon to each of the accounts in the numerator, and add N times epsilon to the divisor such that the row sum still adds up to one.
* In the real-world example, you might not want to apply smoothing to the initial probabilities in the first row of the transition matrix. That's because if you apply smoothing to that row by adding a small value to possibly zeroed valued entries, you'll effectively allow a sentence to start with any parts of speech tag, including punctuation.

Populating the emission matrix:



* Instead of counting pairs of tags, you will now count how often a word is tagged with a specific tag like noun, verb, or the other tag.
* Forward pass
* Backward pass

You will start by initializing two matrices of the same dimension.

best\_probs: Each cell contains the probability of going from one POS tag to a word in the corpus.

best\_paths: A matrix that helps you trace through the best possible path in the corpus.

Coding tips:

* Python's context managers:

with open ("file name", 'r') as f:

lines = f.readlines()

* list comprehension:

words = [line.split('\t')[0] for line in lines]

* defaultdict: they are a special kind of dictionaries that return the "zero" value of a type if you try to access a key that does not exist.

freq = defaultdict(int)

for word in words:

freq [word] + = 1

* Python String Methods:

Such as isdigit(), endswith(), isupper()

* Matrix can be hard to read, as Numpy is more about efficiency, rather than presenting values in a pretty format. For this you can use a Pandas DataFrame. In particular, a function that takes the matrix as input and prints out a pretty version of it will be very useful.
* Since the sum of the rows was desired the axis was set to 1. In Numpy axis=1 refers to the columns, so the sum is done by summing each column of a particular row, for each row.