Tutorial: Markov Chain and PageRank for Keywords Extraction and Ranking from News Articles

Parth Mihir Patel

31/10/2022

1. Introduction

Keywords extraction, that is the process of extracting the most important and relevant words from a text or a set of texts, is one of the core components of every News Aggregator Service. This tutorial elaborates how Markov Chain can be used for extracting nouns from a set of news articles, and then ranking them in order to extract the top five. The process involves noun extraction using POS tagger, construction of a graph of extracted nouns, and PageRank algorithm for ranking nouns.

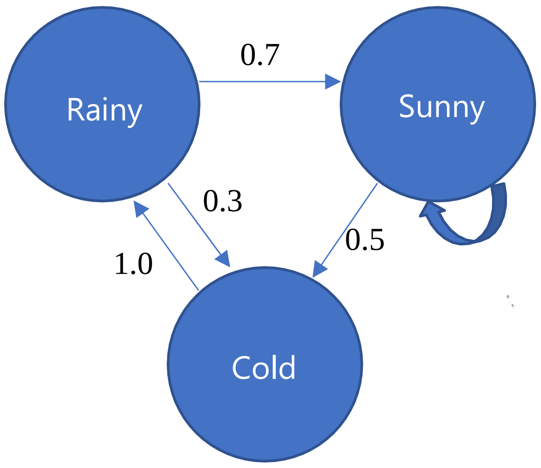
Latest news articles are retrieved using NewsAPI. Their summaries are obtained by feeding each article to GPT-2 model[1]. Nouns are extracted from each summary using Hidden Markov Model based POS tagger. A graph data structure of all the extracted nouns is constructed. Nouns are represented as nodes and nouns belonging to same summaries are connected with edges. Importance of nodes(nouns) is ranked using PageRank[2] algorithm. PageRank was originally invented by Google's founders for the purpose of ranking the web-pages, but as elaborated later in this tutorial, it is very suitable for the purpose of ranking keywords as well.

Both POS tagging and PageRank algorithm are based on Markov Chain.

(1.1) Markov Chain

Markov chain is a mathematical system consisting of states connected by transition probabilities. Transition probabilities are probabilities of hopping from one state to another. Sum of all outgoing transition probabilities from each state has to equal 1.

Example:



(fig 1, example of Markov Chain)

In this example, “Rainy”, “Sunny” and “Cold” are states.

Transition probabilities can be packed in a transition matrix P:

	R	S	C
R	0	0	1
S	0.7	0.5	0
C	0.3	0.5	0

(table 1, transition matrix)

State vector represents the probabilities of being at respective states at t^th timestep/iteration.

Let’s say the initial state vector is [1,0,0] , that is, 100% probability of being at “Rainy” state in the beginning.

To calculate the state vector at next iteration (π_{t+1}), the transition matrix is multiplied with current state vector.

$\boldsymbol{\pi_{t+1} = P * \pi_t}$

After some iterations, the state vector will stop changing.

$\pi = P\pi$

(π will stop changing)

This is called the stationary distribution state vector. It indicates the probabilities of landing at respective states over long run.

(1.2) Hidden Markov Model

It is a type of Markov model in which some states are hidden. For example, if a model has following states – {HighCholestroal, LowCholestrol, Happy, Sad}. Here, Happy and Sad are observable states. Cholestrol level influences the observable states, hence HighCholestrol and LowCholestrol are hidden states.

Probability of hopping form a particular hidden state to another hidden state is called transition probability. Probability of hopping form a particular hidden state to a particular observable state is called emission probability.

2. POS Tagging

Parts of Speech tag means different categories of words, such as noun(N), preposition(P), verb(V), adjective(A), conjunction(C), etc. Finding the Part of Speech tag of every word in sentence/para is called POS tagging.

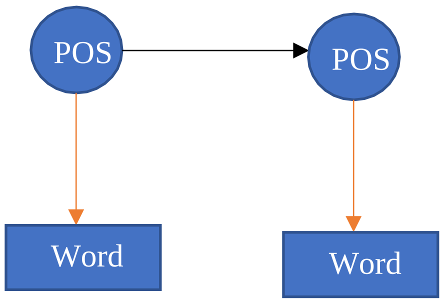
There are various techniques that can be used for POS tagging such as:

1) **Rule-based approach**: Predefined rules for context and position of the words, are used for assigning POS tags
2) **Deep learning models**: Neural networks such as Meta-BiLSTM
3) **Stochastic tagging**: It uses frequency and probability. Simplest approach is to find out themost frequently used POS tag for each word in the training data, and use this information

on unseen data. But this doesn't consider context of the words. For example, the word "watch" might be used as a noun in one context but as a verb in another context. The simple approach doesn't understand this. A more sophisticated stochastic tagging approach uses Hidden Markov Model.

(2.1) Hidden Markov Model for POS Tagging

When using this model for POS tagging, words are considered as observable states and POS are considered as hidden states.



(fig 2, HMM for POS tagging)

Transition probability : Probability that a particular POS tag x will be followed by POS tag y.
$P(POS_{i+1}|POSi)$
For example, the probability that a noun is followed by a verb.

Emission probability : Probability that a particular word will appear given a particular POS tag.
$P(word|POS)$
For example, the probability that the word is "Bill" given that the POS tag is noun.

Transition and emission probabilities need to be learned from training corpus.
Example:
Let's say we have a dummy dataset of these 4 sentences –
"Price of watch soars because of inflation" (N P N V C P N)
"Watch PM's speech on inflation" (V N N P N)
"PM tries to fan investment" (N V P V N)

“Fan price soars” (N N V)

First create a table of number of times a POS tag x is followed by a POS tag y (Like noun is followed by verb twice in the above dataset):

	N	V	P	C
<S>	3	1	0	0
N	2	2	2	0
V	2	0	1	1
P	3	1	0	0
C	0	0	1	0

(<S> indicates start)

Now divide each cell by corresponding row total. These would be transition probabilities. Total of each row will be equal to 1.

(table 2, transition frequency table)

To calculate emission probabilities, prepare table of frequencies of each word w.r.t. each POS tag. (Number of times Watch appears as noun, number of times Watch appears as verb, number of times PM appears as noun, etc).

Emission frequency table has been shown on the next page.

	Price	Of	Watch	Soar	Because	Inflation	PM	Speech	On	Try	To	Fan	Investment
N	2	0	1	0	0	2	1	1	0	0	0	1	1
V	0	0	1	2	0	0	0	0	0	1	0	1	0
P	0	2	0	0	0	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	0	0	0	0	0	0	0

(table 3, emission frequency table)

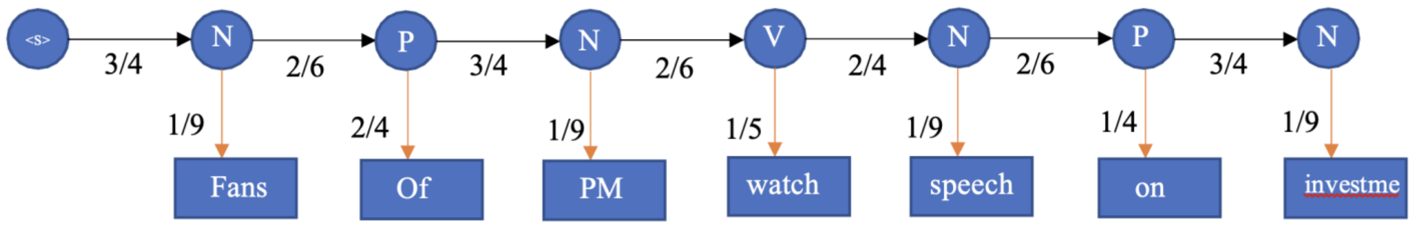
Now divide each cell by corresponding row total. These would be emission probabilities.

In real life, the training corpus will be huge and contain almost all words in English. For example, Wikipedia articles dataset.
Once we have transition and emission frequencies, we can find out POS tags for tokens in any sentence (as long as every word in that sentence was in training corpus).

Example:
"Fans of PM watch speech on investment"
Since each word could be one of the 4 POS tags, and since this sentence has 7 words, there are 4^7 different combinations of POS tag sequences.
Joint probability for each of the possible sequences would be calculated.

Let's consider joint probability of one of the possible sequences –

(next page)



(fig 3, a possible sequence)

= 1/9 * 2/6 * 2/4 * 3/4 * 1/9 * 2/6 * 1/5 * 2/4 * 1/9 * 2/6 * 1/4 * 3/4 * 1/9

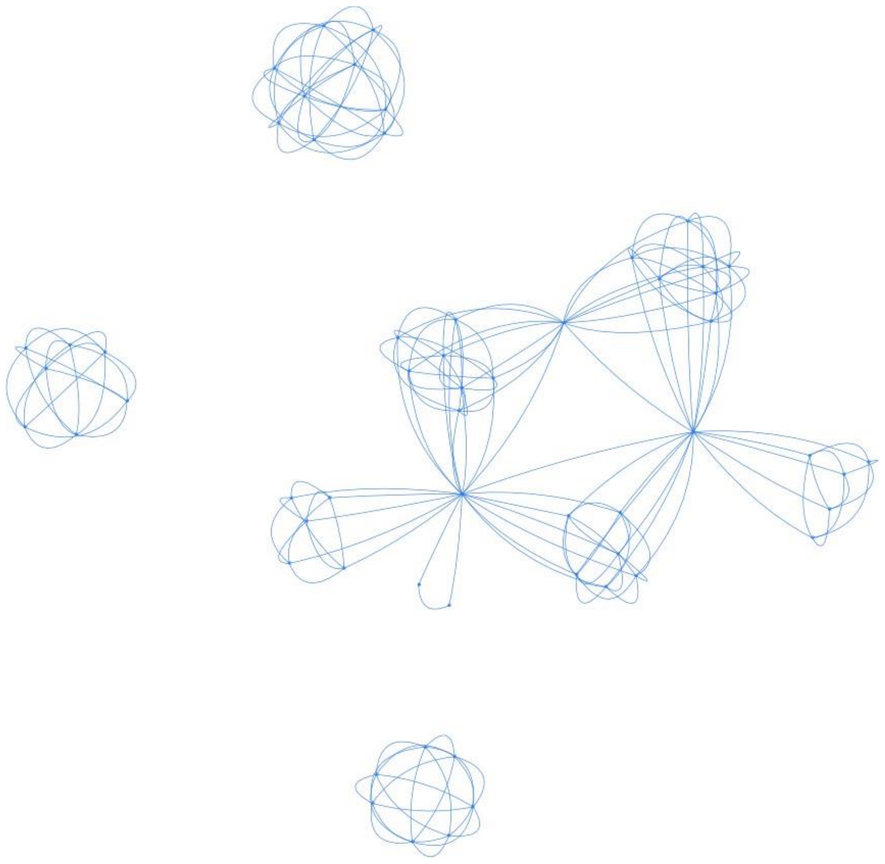
Similarly joint probabilities for all other possible sequences are calculated. POS sequence that has the highest join probability will be used for assigning tags.

Once POS tagging is done for all sentences in all article summaries, nouns are extracted from each summary.

3. PageRank Algorithm

To find the most important nouns, firstly a Graph data structure has to be constructed. Nouns are represented as nodes. Nouns belonging to the same article summaries are connected to each other with edge. In case multiple article summaries share the same noun, that noun would act as a bridge between graph components. To construct the graph from nodelist and edgelist, networkx library for Python can be used.

Visualized graph has been shown on the next page.



(fig 4, graph of nouns visualized using pyvis library for python)

This graph can be considered as a Markov Chain. Nodes can be considered states. The probabilities of landing at a particular state(node) over long run is obtained in stationary state vector.
Most important nodes(nouns) would be the ones on which there is the highest probability that we would end up there over a long run in a random walk. Hence we need to obtain stationary state vector, sort its entries in ascending order and get the first 5 entries.

For the initial state vector, let's consider that the state vector is:

$$\pi = (1/n\ ,\ 1/n\ ,\ \ldots\ ,\ 1/n)$$

where $\boldsymbol{n}$ is the total number of nodes. This means equal probability of being at every state(node).

(3.1) Transition matrix

Dimension of it will be n * n, where n is the number of states(nodes). For every column i (i >=1 and <= n), all entries of that column will be equal to 1/k where k is the number of outgoing edges(transitions) from that node(state).

Now the issue here is that some nodes may have no outgoing edges, meaning that the probability of transitioning from that node to any other node would be 0. In such a case, we would be stuck at that node and our algorithm wouldn't work. This is called spider trap problem[4]. To fix this, the concepts of teleportation and damping factor need to be introduced.

(3.2) Teleportation and Damping Factor

Connection (transition) is established between each node (state) of chain with every other node (state). The edges (transitions) that didn't exist in original chain, will have very low transition probabilities. This is called teleportation. In case if a node originally had no outgoing edges(transitions), teleportation makes sure that we don't get stuck at that node, and that we hop over from it randomly to any other node (state) in thechain. The transition matrix after introducing teleportation will be:

$$R = b * P + ((1 - b) * v * e^T)$$

Where P = original transition matrix,

$e^T = (1/n , \ldots\ldots, 1/n)$

$v = (1, 1, \ldots, 1)^T$

b is the damping factor. It is usually set to 0.85.

All the entries that were zero in matrix P will become (1-b\n) in this new matrix. Sum of all entries in each column will still be 1.

(3.3) Stationary distribution and Keyword extraction

At every step, the random walker will jump to another node according to the transition matrix. After some timesteps, the state vector would stop changing. That state vector would be the stationary distribution.

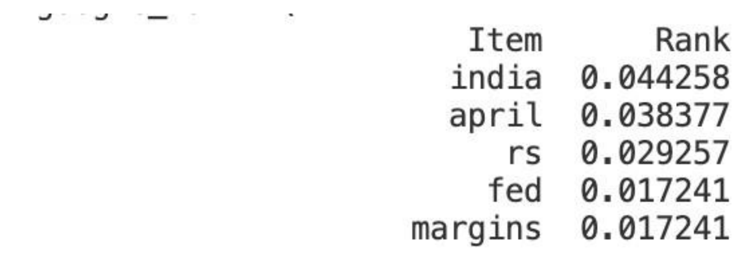
$\pi = R\pi$

This equation(if we rearrange it as ***Rπ = 1 * π***) resembles the equation of eigenvector which has eigen value of 1. Hence the stationary distribution state vector can be obtained by calculating the eigen vector for matrix R, which has eigen value of 1.

Eigen vector is a vector which doesn't rotate in the vector space after multiplying it with a particular matrix. It just scales by a scalar, and that amount is called eigen value.

To find eigen vector corresponding to R, eigen value (1 in this case) is subtracted from its diagonals. Let's call this new matrix R_{new}. And then this equation is solved to obtain ***π*** :
R_{new} * ***π*** = null vector

We would sort the probabilities in ***π**, **and the top 5 nodes with the highest probabilities will be top 5 keywords.***



```
   Item      Rank
  india  0.044258
  april  0.038377
     rs  0.029257
    fed  0.017241
margins  0.017241
```

(fig 5, PageRank's output)

4. Conclusion

The tutorial has demonstrated how the simple concept of Markov Chain (on which both - Hidden Markov Model based POS tagger, and PageRank algorithm, are based) is enough to extract keywords from news articles. This approach is more powerful than rules-based approach and frequency based simple stochastic approach, and it is much simpler to implement than deep learning based approach.

References

[1] – Alec Radford et al, 2018 “Language Models are Unsupervised Multitask Learners”

[2] – Sergey Brin, Lawrence Page, 1998 “The anatomy of a large-scale hypertextual Web search engine”