20-PBD-001
PARTH M PATEL
MSC BDA SEM-4 PROJECT

Financial news summarization, keywords extraction, and recommendation

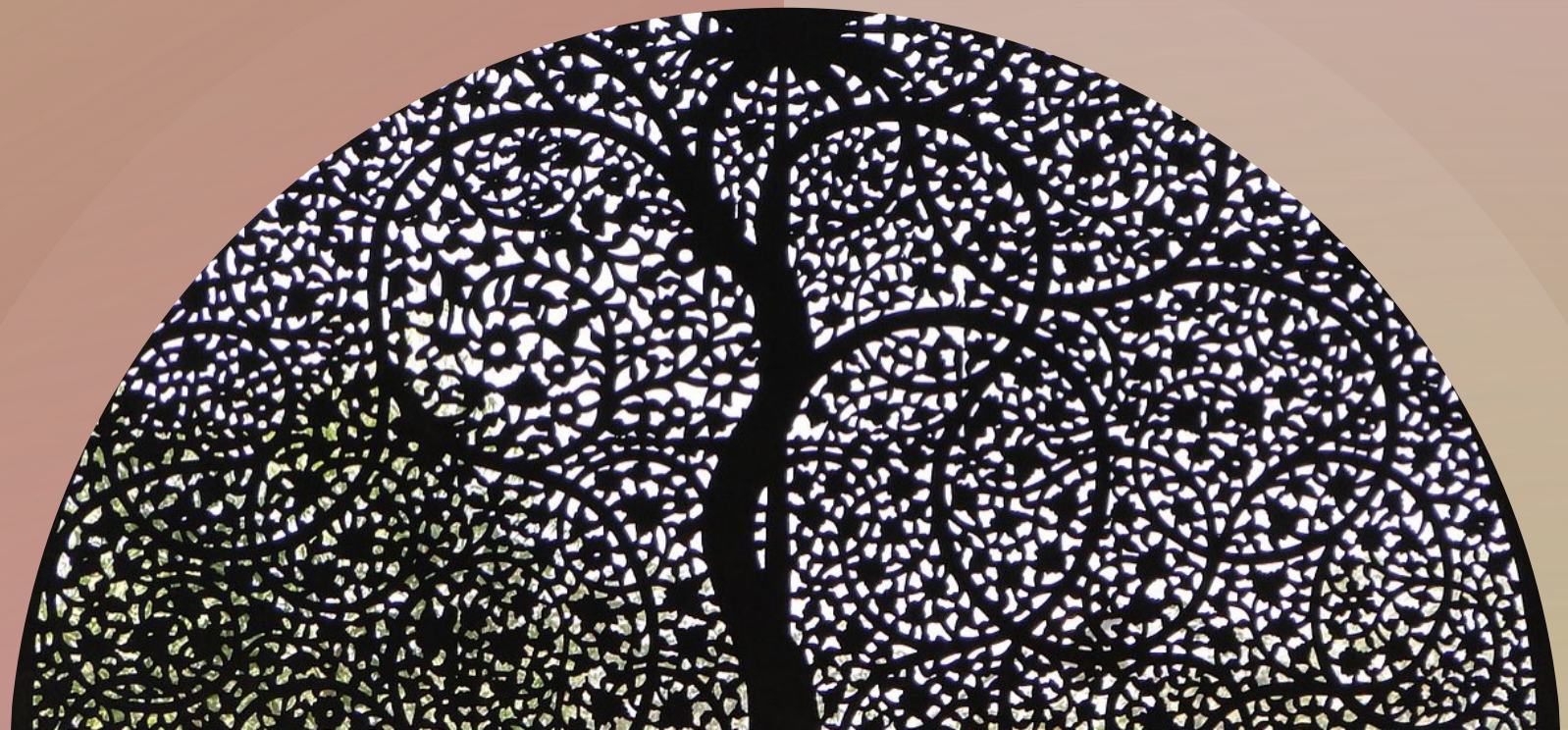
PURPOSE

- Most news summarization apps currently available on the PlayStore/AppStore use LSTM architecture, which has some flaws when it comes to dealing with large lengths of texts such as news articles. To deal with quality issues, those apps often deploy human curators for curating the outputs of LSTM.
- Transformer architecture is one of the latest breakthroughs in the field of NLP, and it addresses the flaws of LSTM. This project makes use of this cutting edge architecture to do summarization of Financial News Articles, and presents those summaries in an elegant way to the user in the app.
- Besides, the app also provides functionalities such as being able to view articles based on “Top Keywords” , and recommending unread articles to the user.

ABOUT THE PROJECT

- The project is divided into 3 modules – Summarization, Keywords Extraction, and Recommendation.
- Transformer neural network architecture introduced by Ashish Vaswani in August 2017, has been used for summarization. Each article is summarized in 100 words.
- The second module of this project focuses on keyword extraction. Top 5 most important words will be extracted from the summaries that we have obtained (not 5 keywords from each summary, but 5 from all summaries in total). User would be able to get News articles(their summaries) which contain those keywords. Markov Chain and PageRank algorithm are the crux of this module.
- The third module of this project is a recommender system which recommends unread articles to the user based on what they've previously read. The concepts of TF-IDF, dimensionality reduction using SVD, and cosine similarity, have been used for this module.

INFRASTRUCTURE



DATABASE TECHNOLOGIES

NOSQL

- Alternative to relational database systems
- In RDBMS, entities are modelled as tables with predefined schema, whereas in NoSQL, they are modelled as JSON documents
- Each Document contains key value pairs.
- Schema isn't predefined, hence can store unstructured data
- Documents can store arrays and objects, hence no need for JOIN operations unlike RDBMS

COSMOSDB

CosmosDB is a cloud NoSQL database service provided by Microsoft Azure

It is a distributed database which can scale horizontally

After creating a CosmosDB container on Azure Portal, we can interact with CosmosDB using CosmosDB's python API

For this project, I have the created following containers in CosmosDB:

Summaries container
For storing article summaries. The schema of each document in this container will be:
isRead, Title, URL, Summary, Image.

Keywords container
For storing summaries of articles which contain specific keywords. The schema of each document in this container will be:
Keyword, isRead, Title, URL, Summary, Image.

Recommendations container
For storing summaries of articles which are recommended to the user. Schema same as Summaries container.

BACKEND TECHNOLOGIES

WEB API

A set of functions and routines which can be accessed via the HTTP protocol by the developer, to be used for their application.

Web API is hosted on a server. When client makes a request to the server using predefined endpoint, the server returns response in XML or JSON format.

In this project, I've created Web API for returning the summaries stored in our CosmosDB, as JSON.

Our frontend app will make GET request to the API, retrieve the JSON text, and then deserialize it and display it to the user.

For building the API, I've used Flask framework.

My Web API has following endpoints:

/getSummaries

/markRead{id}

/getRecommends

AZURE APP SERVICE

Hosting a web API on our own private server is expensive and not scalable. Hence we instead use cloud hosting services from platforms such as Microsoft Azure. Azure's cloud hosting service is called Azure App Service.

While creating app service on Azure Portal, we can specify host's operating system, host's RAM, host's processor, etc.. We can change these things anytime as we scale up. Once the App Service has been created, we deploy our web app on it.

It has robust security features such as IP whitelisting, IP blocking, etc..

AZURE FUNCTIONS

Microsoft Azure's serverless computing service is called Azure Functions

Serverless means the developer doesn't have to specify host's RAM, CPU etc

The program deployed on Azure Function can be triggered based on events

GLOSSARY OF LIBRARIES AND API'S

FLASK

- A Python framework for building web applications, REST APIs, handling HTTP requests, returning JSON response, rendering pages, etc

NEWSAPI

- A REST API that returns latest news stories in JSON format. Note that it doesn't return entire articles, just their details such as names, dates and links.
- Two endpoints:
  newsapi.org/v2/everything/ - This endpoint gives new and old news from a repository for 80k articles
  newsapi.org/v2/top-headlines/ - This endpoint gives latest breaking news for the country
- The API can be called using *requests* library of python. We need to specify endpoint URL, and, pass APIKey parameter.

Following parameters can be passed in the URL:

- **q:** Used for specifying list of keywords
- **category:** The category we want to get headlines of
- **sources:** The websites/blogs/channels we want to get out news from
- **totalResults:** Number of articles returned
- **articles:** Array of articles objects
- **url:** Link to article
- **titlepublishedAT:** Date at which the article was published

The response, which is a JSON document, has following variables:

- **totalResults:** Number of articles returned
- **status:** 0 if query was successful
- **articles:** Array of articles objects

An article object contains:

- **url:** Link to article
- **Title**
- **publishedAT:** Date at which the article was published.

NEWSPAPER LIBRARY

Web scraping is the process of extracting relevant text out of webpages

It is a tedious process which requires parsing HTML tags. Parser also needs to be context sensitive.

Python has a library called “newspaper3k” for web scraping

It uses advance algorithms with web scraping to extract all the useful text from a website

PYTORCH

PyTorch is a popular deep learning library

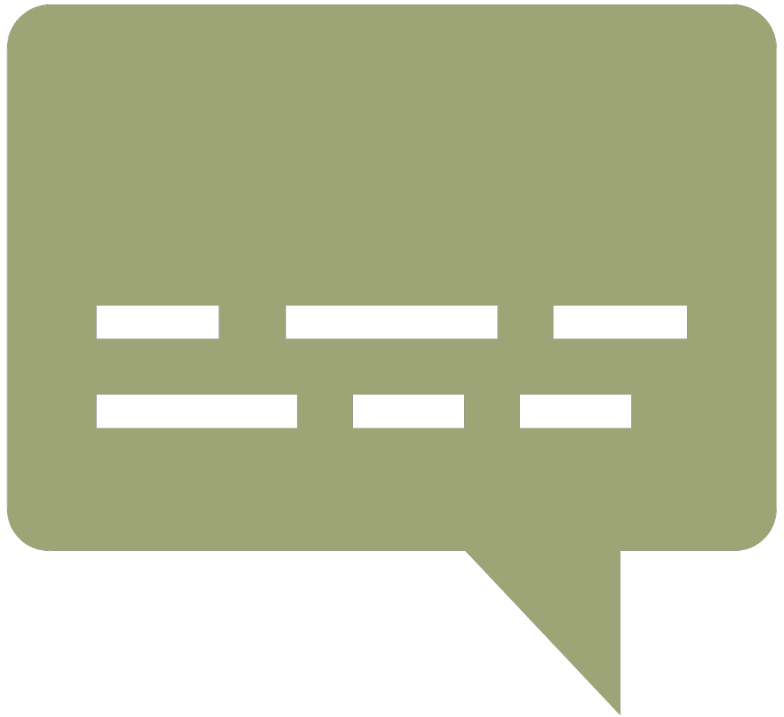
The project uses it for building the neural network model, training it, and calculating the accuracy

torch.transformers is a sub library of PyTorch which provides transformer neural network models like GPT and BERT

TEXTBLOB, SPACY AND NLTK

Libraries for text preprocessing.

- Tokenization
- Spelling Correction
- Lemmatization
- POS Tagging
- Word Frequencies
- Sentiment Analysis
- n-grams



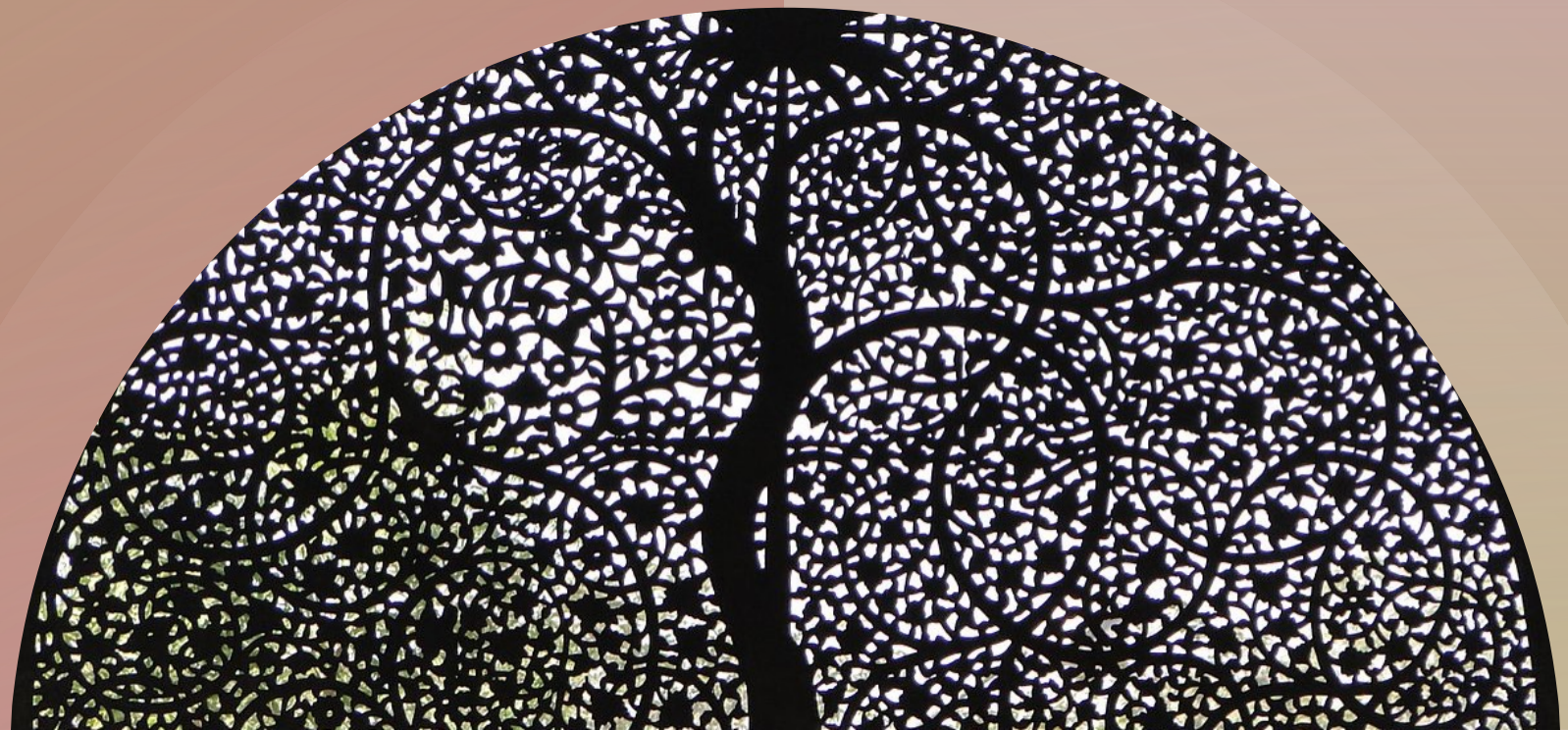
NETWORKX

A library for constructing graph data structure from a list of nodes and edges

It provides important graph algorithms such as breadth first search, depth first search, Kruskal's, Prim's, etc

It also provides network analysis capabilities, such as degree centrality, Betweenness, closeness, community detection, etc

SUMMARIZATION



SYSTEM DESIGN OVERVIEW

TRAINING PHASE

Load	Preprocess	Train	Evaluate
Load BBC News Dataset into pandas dataframe	Perform text preprocessing and encode both ArticleText and Summary columns	Train GPT-2 model on data	Evaluate the trained model

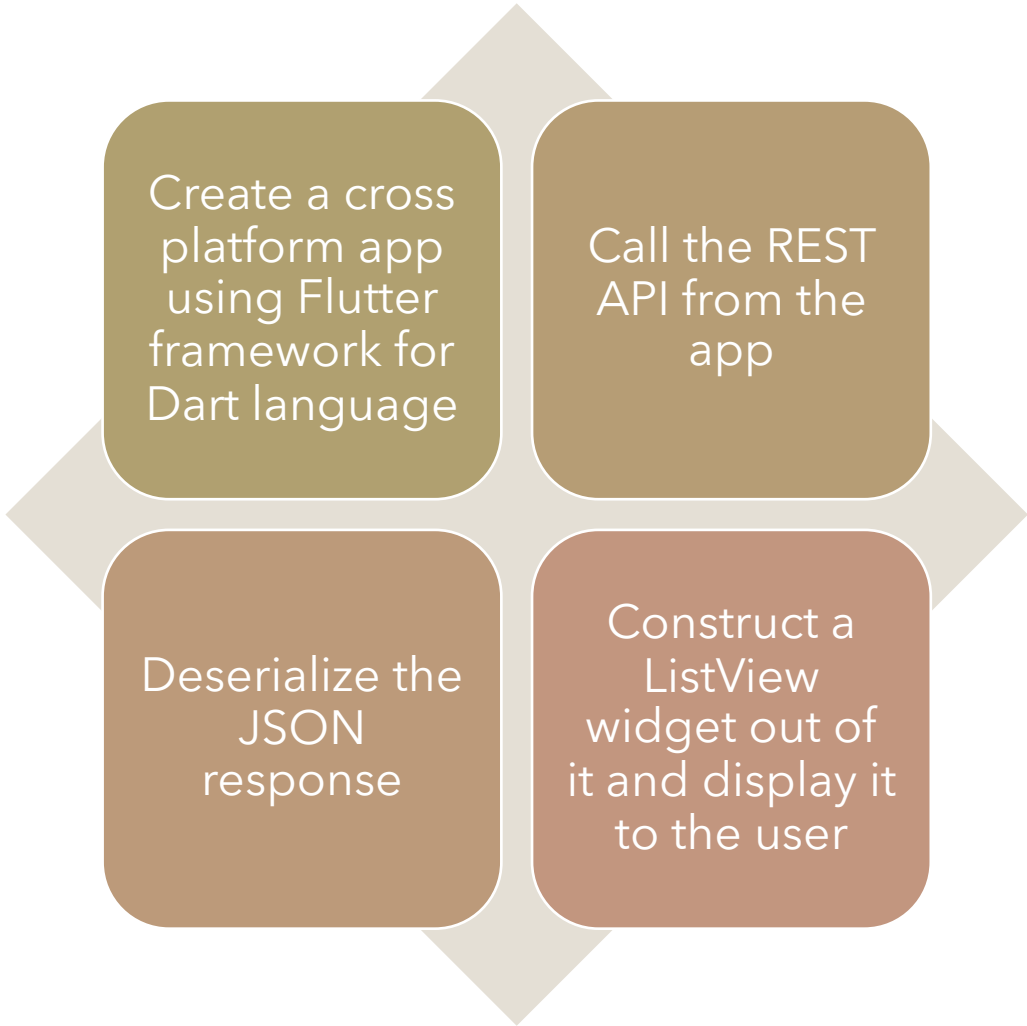
DEPLOYMENT PHASE

- Load latest news using NewsAPI
- Deserialize the JSON into a collection of Article objects
- Iterate through each Article objects, get the link variable, perform web scraping on that link using newspaper library
- Feed the scraped text into model and obtain summaries
- Use CosmosDB API to insert summaries into our “Summaries” container on Azure CosmosDB

REST API

- Create a REST API using Flask framework
- Create a GET request endpoint /getSummaries
- /getSummaires will fetch the documents stored in Summaries container of Azure Cosmos Database using CosmosDB library, parse them, and return them in JSON format
- Deploy this REST API on Azure App Service

FRONTEND


Create a cross platform app using Flutter framework for Dart language
Call the REST API from the app
Deserialize the JSON response
Construct a ListView widget out of it and display it to the user

SUMMARIZATION TECHNIQUES

EXTRACTIVE

- Extractive summarization is similar to how we highlight lines on paper
- It calculates the importance of sentences in an article, and extracts them
- It does not construct any new words or phrases

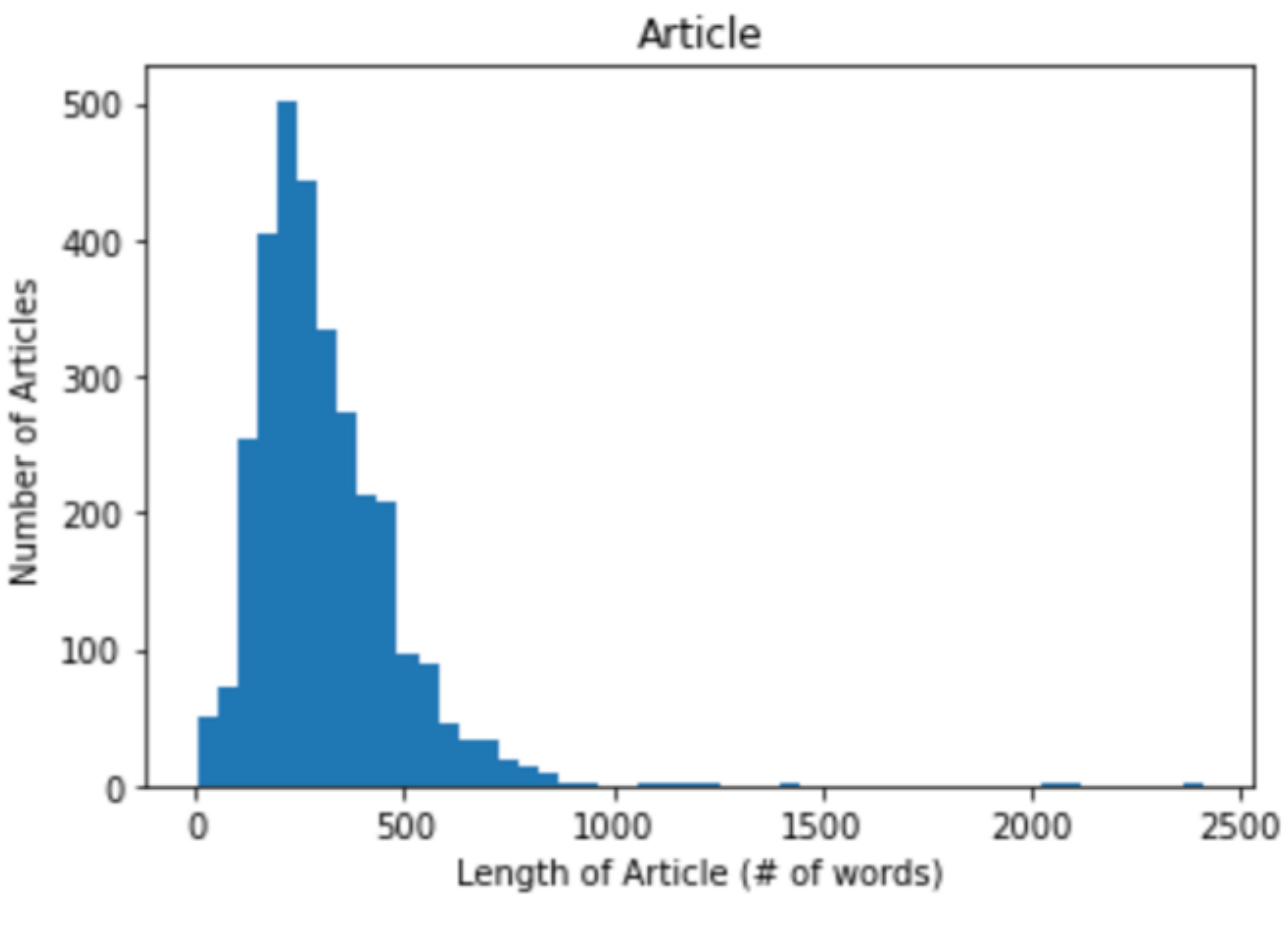
ABSTRACTIVE

- Abstractive summarization tries to understand meaning/point of each sentence and context of each word, in the article
- Based on that, it creates new phrases in a meaningful way, such that they carry the same meaning in a rephrased shorter way
- It is more complex than extractive summarization

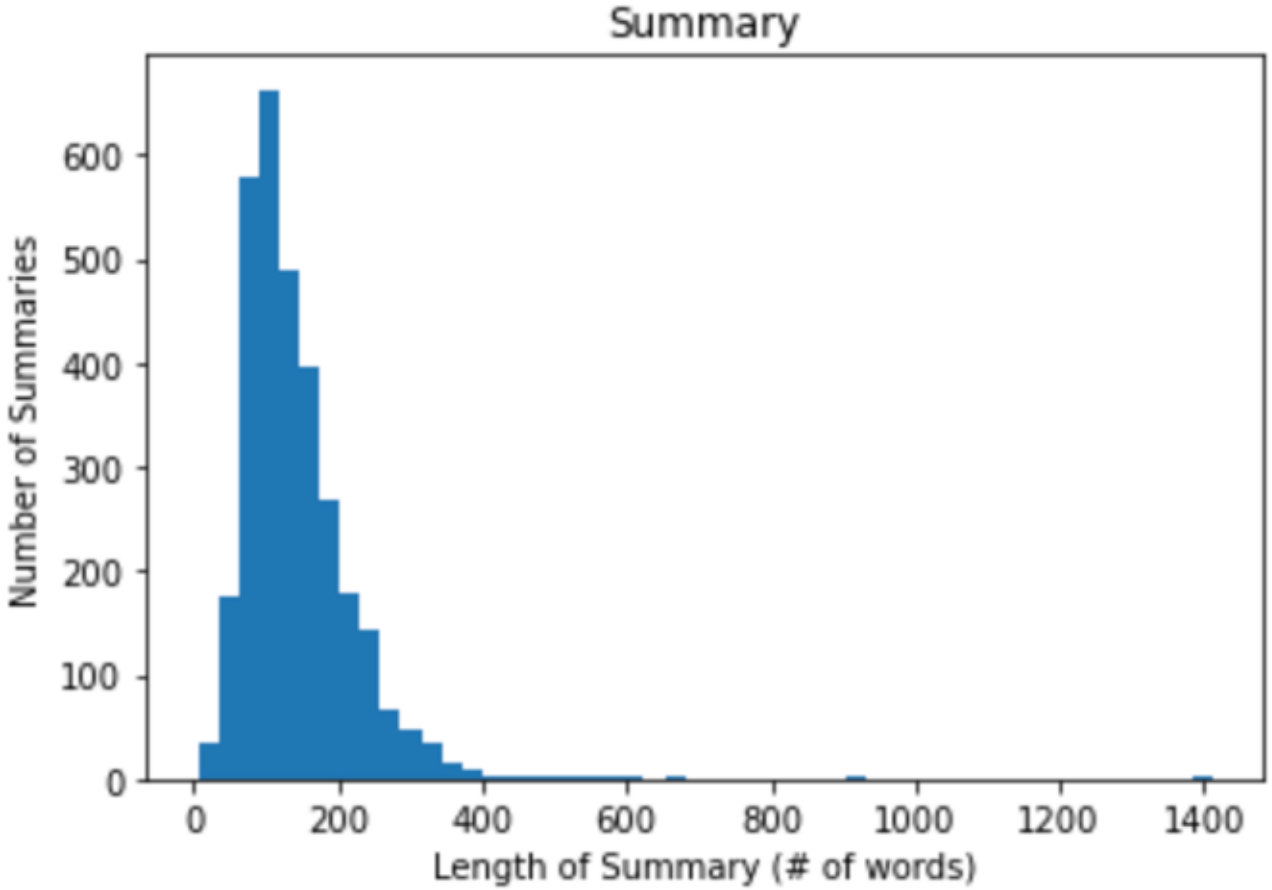
DATASET OVERVIEW

BBC News Dataset is a csv with 2224 tuples

It has 3 features - ArticleText, Class and Summary



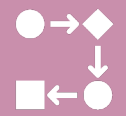
THE LENGTH(WORD-COUNT) OF ARTICLE TEXTS IS DISTRIBUTED AS FOLLOWS



SUMMARY WILL BE THE TARGET VARIABLE. LENGTH(WORD-COUNT) OF SUMMARIES IS DISTRIBUTED AS FOLLOWS

TEXT PREPROCESSING

TOKENIZATION



Tokenization is the first step in pre-processing

It is the process of breaking paragraphs of texts into discrete elements

If we want our discrete elements to be words, we use whitespace as the delimiter, and if we want sentences, we use stopmark

TOKENIZATION

In the project I've used *spacy.tokenizer.Tokenizer* for tokenization.

```
from spacy.tokenizer import Tokenizer
from spacy.lang.en import English
lang = English() #Load language
tokenizer = Tokenizer(lang.vocab)#Create tokenizer with Eng vocab
article_tokens = tokenizer(articleText)
```

STEMMING/LEMMATIZATION

Stemming and Lemmatization are processes for removing inflections from tokens, thereby reducing words to their base forms

Stemming uses unsophisticated heuristic methods to simply chop off the ends of tokens

Lemmatization performs advanced vocabulary and morphological analysis of words

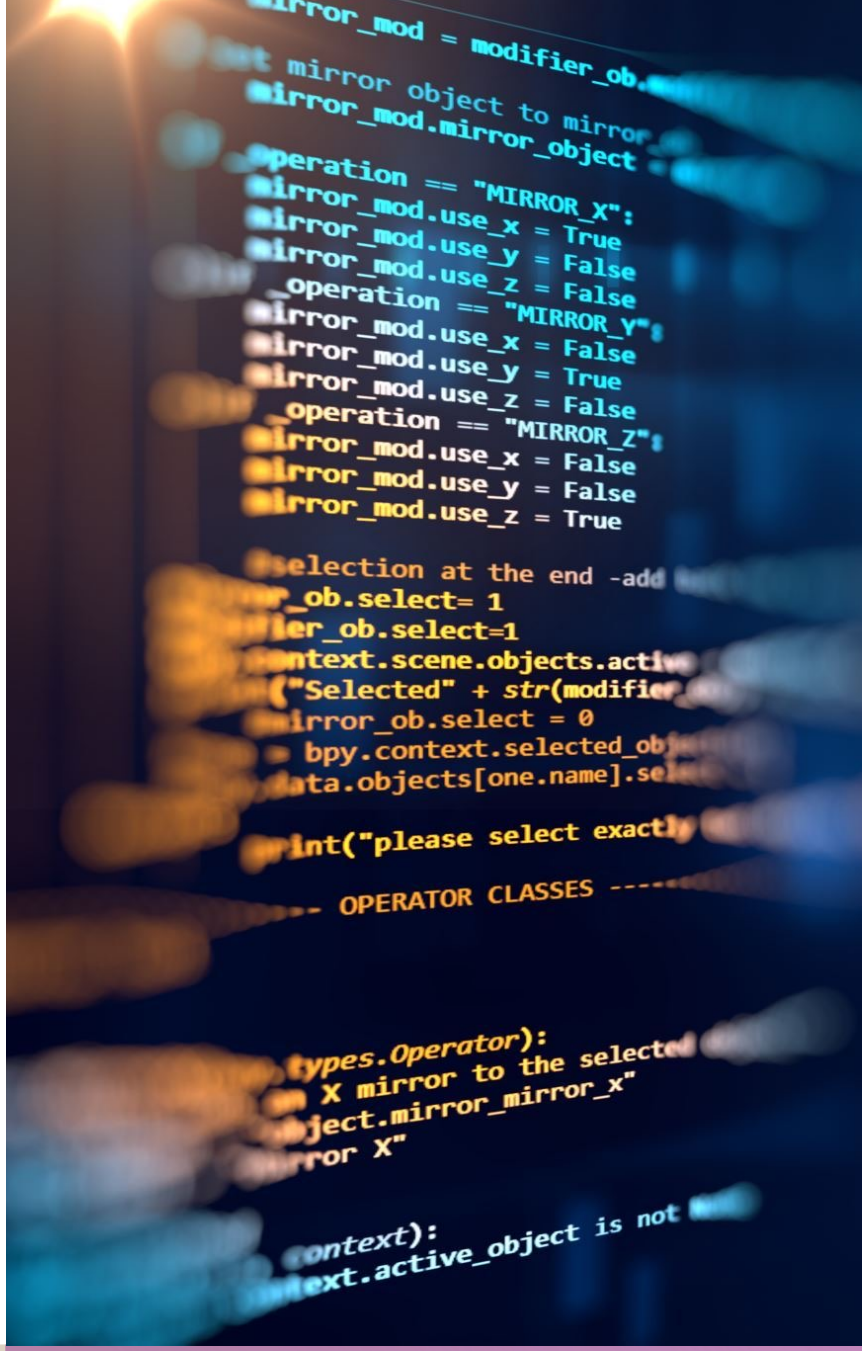
LEMMATIZATION

In the project I've used nltk.stem.WordNetLemmatizer for lemmatization.

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

lemmatized_article_tokens = []

for tok in article_tokens:

        lemmatized_article_tokens.append(lemmatizer.lemmatize(tok))
```

ENCODING

- Words are categorical data
- Since we can’t feed categorical data into our neural network, we need to convert each word into a vector
- One Hot Encoding is the simplest way to do this
- Each word is represented with a vector of size n, where n is number of words in our vocabulary



PROBLEMS WITH ONE HIT ENCODING

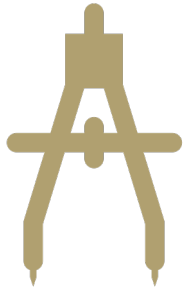
The dimension of our vectors is as big is size of our vocabulary

Vectors aren't contextualised

WORD2VEC

Word2Vec algorithms create vectors(embeddings) that are Contextualised

Word2Vec helps establish a word's association with other words



Vectors of words that have similar meanings, have small cosine distance in the vector space

CBOW

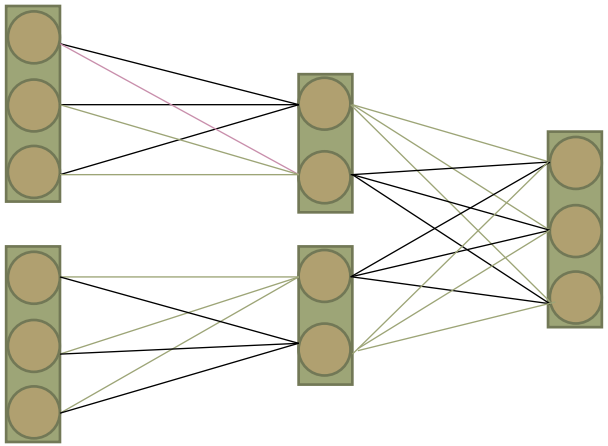
CBOW IS AN ALGORITHM FOR WORD2VEC

It is based on a neural network which tries to predict the one hot encoding of target word based on one hot encodings of its surrounding words.

Consider the sentence ‘GDP is on big decline’. The model converts this sentence into word pairs in the form (contextword, targetword). The word pairs would look like this: ([GDP, on], is), ([is, big], on),([on, decline], big).

With these word pairs, the model tries to predict the target word considered the context words.

CBOW



The input layer will be One Hot Encodings of 2 context vectors. Each of these will be connected to a hidden layer with k neurons each (k = 2 in the above figure). Note that these two hidden layers will be parallel and will share the same weights.

These 2 parallel hidden layers will be concatenated and fed into the output dense layer which has softmax activation and n neurons where n is the size of our vocabulary (n = 3 in the above figure).

As we train this network on all the word pairs, the weights of the hidden layer will be learned, and the hidden layers will correspond to our learned word2vec embeddings. Embeddings will have k dimensions.

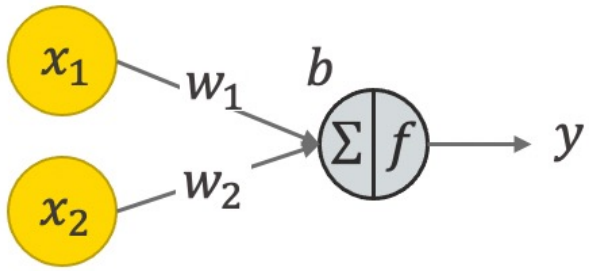
WORD2VEC

In the project I've used torch.nn.Embedding for creating embeddings.

```
from torch.nn import Embedding

embeddings = Embedding(vocab_size, embedding_size)
```

MODELLING

SINGLE NEURON



Let's say there are n features. We multiply each feature by a corresponding weight, and add them up. The resultant value is called pre-activation value.

$h = w_1x_1 + w_2x_2 + \ldots + w_nx_n$

Now, we pass h through an activation function, let's say Sigmoid. It would squash the pre-activation value between 0 and 1, essentially giving us a probability score.

$a = 1 / (1+e^h)$

NEED OF ACTIVATION FUNCTION

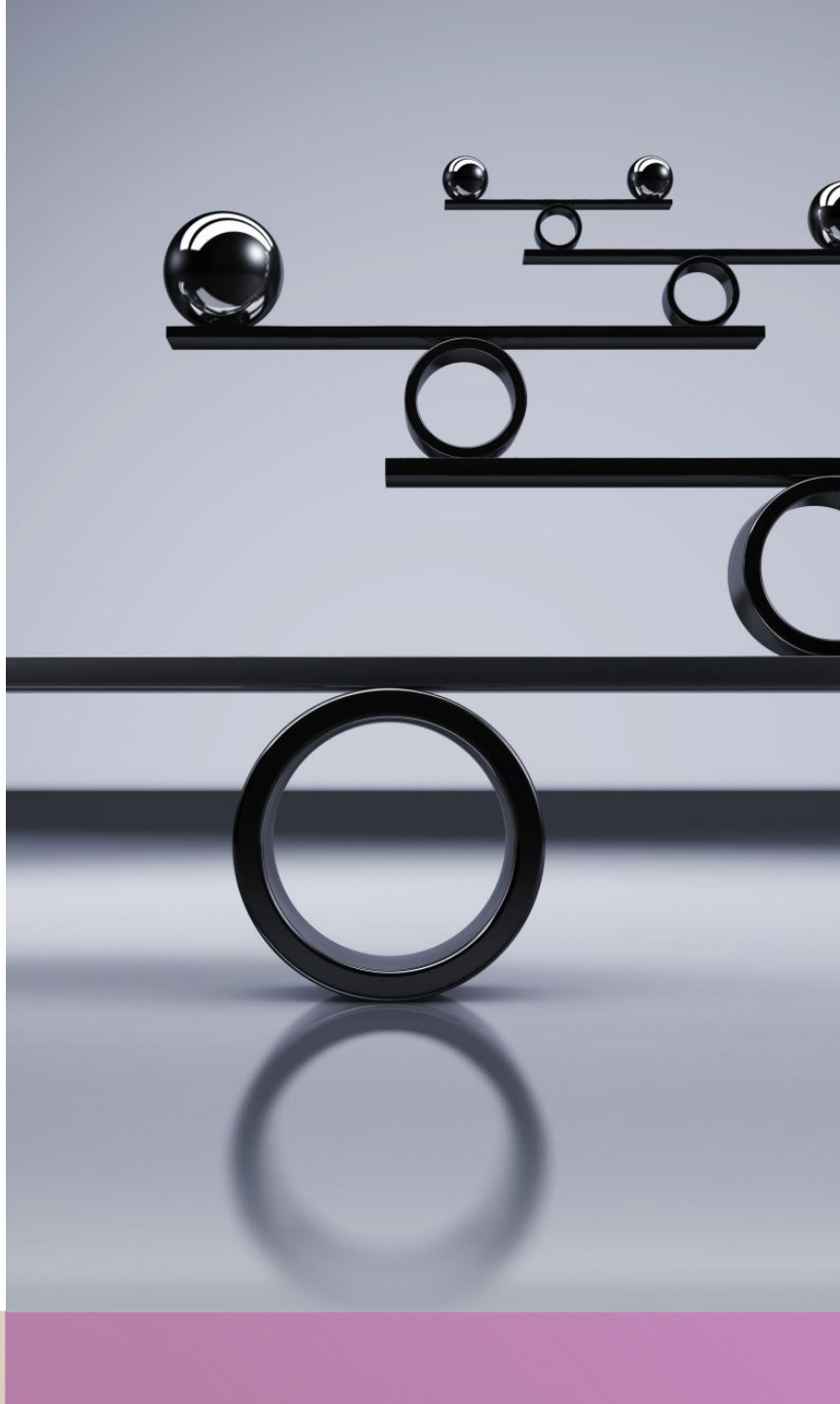
To prevent super large outputs

To map non-linearities

GRADIENT DESCENT

How do we decide the weights such that our neuron is able to map a relationship as accurately as possible? We learn them from training data.

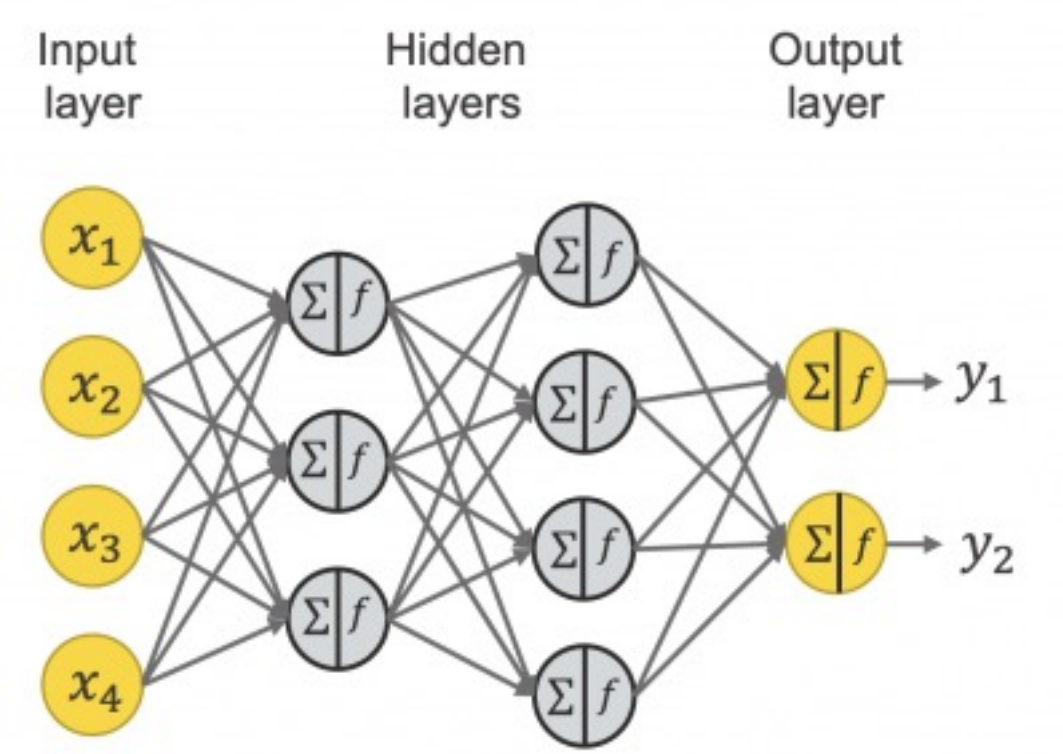
- To begin with, we randomly initialize our weights
- Then we feed a training tuple's independent variables into our neuron. We compare the predicted output of neuron with the real target variable in our training tuple.
- We calculate squared error loss by $(y - ybar)^2$.
- Take the partial derivative of loss wrt each weight parameter using chain rule, and subtract them from weights.
- This process is repeated for n epochs.



NEURAL NETWORK

A single neuron can't map extremely non-linear relationships. To achieve this, we need Neural Network.

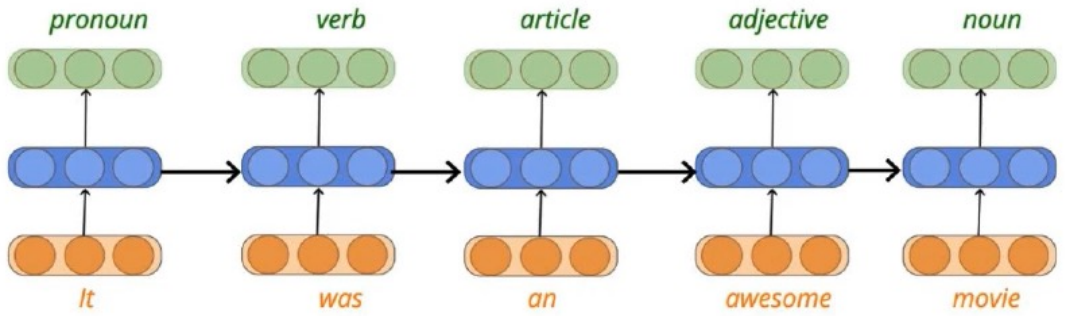
- There are multiple neurons in the 1st layer
- Each neuron in hidden layer has a weighted connection to every neuron in previous layer
- In the output layer, softmax activation is used to get the probability distribution of predicted classes
- The loss is calculated by comparing predicted probabilities with ground truths in training dataset
- The partial derivative is taken with every weight in every neuron using chain rule, and subtracted from current weight



RECURRENT NEURAL NETWORK

When we are dealing with a sequence of inputs (such as a sentence), we need to use recurrent neural networks.

RNNs help us generate hidden representations of words that are context sensitive



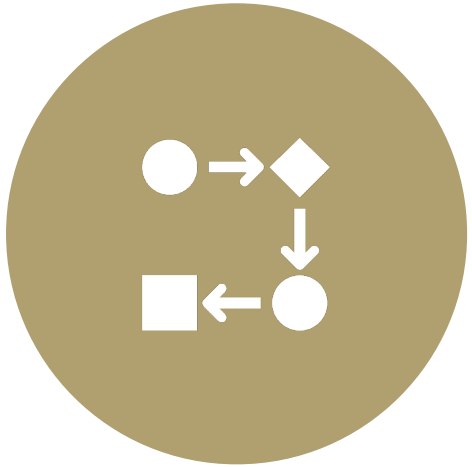
RECURRENT NEURAL NETWORK

RNNs help us generate hidden representations of words that are context sensitive

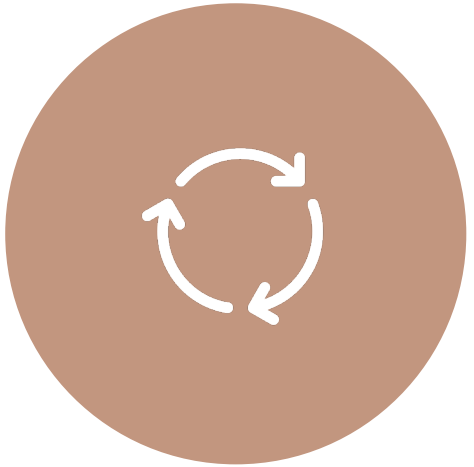
Network's output generated for the word "Bank" needs to be different for "Money in bank", than for "Kids at river bank", even though it is the same word

To achieve this, in RNNs, we iteratively feed the embedding of the current word of sentence as input, but the current iteration's hidden layer is also connected to the previous iteration's hidden layer

PROBLEMS WITH RNN



IT WORKS ITERATIVELY, WORD BY WORD. CAN'T FEED WHOLE SENTENCE AT ONCE



CAN'T MODEL A FUTURE WORD'S INFLUENCE ON THE CURRENT WORD, AS IT IS SEQUENTIAL

ATTENTION MECHANISM

To solve these problems, Ashish Vaswani introduced Transformer model in 2018, in a paper called “Attention is all you need”. It uses the concept of Self Attention Mechanism.

We know the kinds of embeddings that word2vec produces. When the vectors(embeddings) of words that have similar meanings are dot multiplied, the resultant magnitude would be greater than if the words were dissimilar.

We can use this fact to calculate weights and contextualise all words in our input.

- Let's say we have 3 vectors, A, B and C. We want to contextualize each of them.
- Let's first contextualize the vector A.
- -Take the dot product of A wrt A ($A.A = w_1$), dot product of A wrt B ($A.B = w_2$), and wrt C ($A.C = w_3$)
- -Normalize w_1, w_2 and w_2 such that $w_1 + w_2 + w_3 = 1$
- -Now, $A_{new} = A*w_1 + B*w_2 + C*w_3$
- This process will be repeated for other vectors as well. In the end we will obtain contextualised vectors $A_{new,}$ B_{new} and C_{new}

The thing here is, as we can see, the weights (w_n) are pre-calculated, and don't require to be learned during network training process. That's why it is called "self" attention.

But, to make our model more flexible and sophisticated, we need to introduce some weights in the system which are actually learned during training.

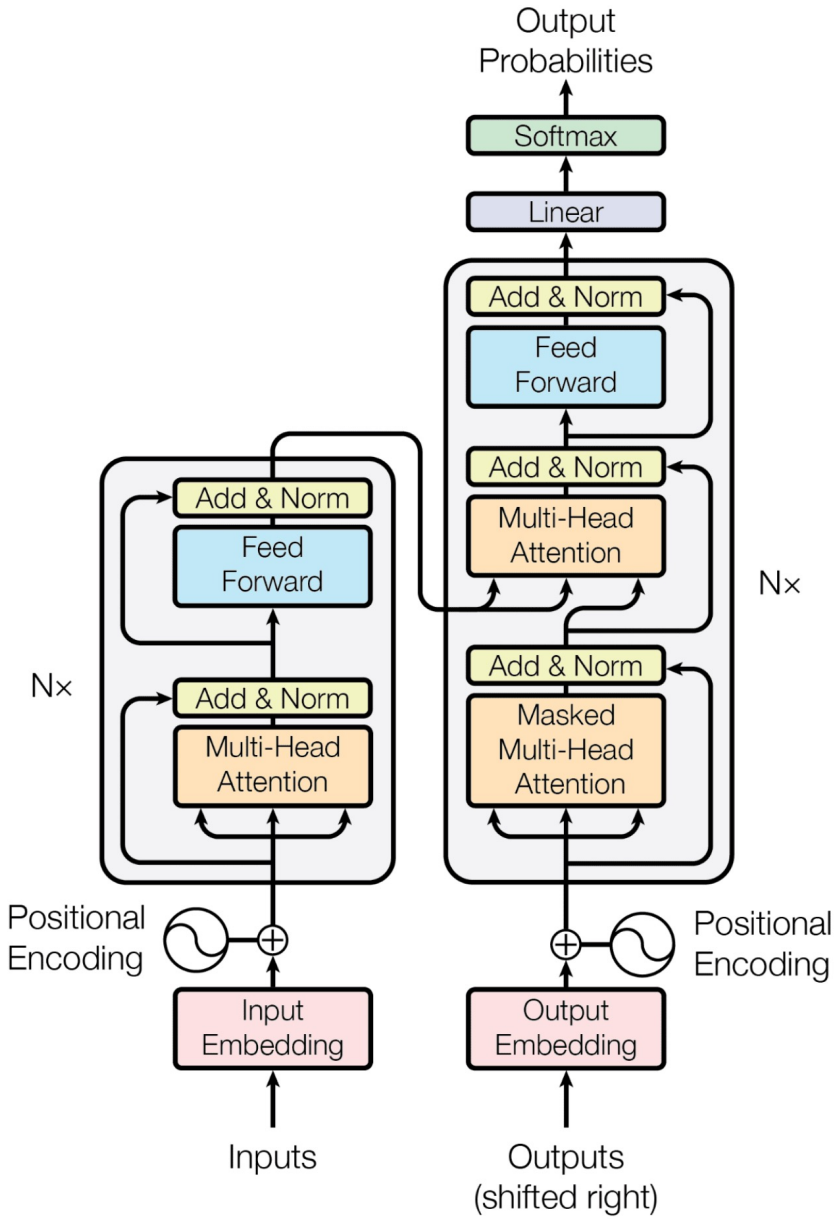
Let's introduce 3 weight matrices – Query(Q), Key(K) and Value(V). Each will have dimension k * k, where k is the dimension of embeddings. Now the updated process for obtaining contextualised vectors will be like this -

-Take the dot product of A wrt A (Q.A.K.A = w_1), dot product of A wrt B (Q.A.K.B = w_2), and wrt C (Q.A.K.C = w_3)

-Normalize w_1, w_2 and w_2 such that $w_1 + w_2 + w_3 = 1$

-$A_{new} = V*A*w_1 + V*B*w_2 + V*C*w_3$

TRANSFORMER MODEL



ENCODER

- Its job is to contextualise all the input embeddings.
- In the input, we feed entire text (embeddings of all words in text) all at once.
- Since the embeddings don't have positional information, we add positional vectors to them. That's because positional information is important for contextualisation.
- The original "Attention is all you need" paper uses this formula to get positional vectors for each position:

$$p_{i,j} = \begin{cases} \sin\left(\frac{i}{10000^{\frac{j}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{\frac{j-1}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is odd} \end{cases}$$

ENCODER

- Positional vectors are added to embeddings, hence we obtain positional embeddings.
- Next, pass positional embeddings though an Attention Mechanism Block. It will output contextualised embeddings.
- Next, we pass these contextual embeddings to a fully connected layer whose neurons use ReLU activation.

DECODER

For i iterations, where i = number of words that we want in our summary,

- In the input, we feed **target** variable's entire text(embeddings of all words in text) all at once. We compute positional embeddings the way that we saw in encoder.
- If we are currently at k^{th} iteration, we keep only first k positional embeddings and mask out the rest.
- We pass these embeddings through attention mechanism block. It will give us contextualised embeddings.
- The next attention mechanism block will computationally combine the contextualised embeddings from previous attention block in decoder, as well as the contextualized embeddings of encoder.
- We pass the contextual embeddings outputted by it, to a fully connected layer whose neurons use softmax activation. This layer will have n neurons were n = number of words in our vocabulary.
- The word corresponding to the output layer's neuron which has the highest value, will be appended to our summary.

TRAINING

TRAINING

Of the 2224 tuples in our dataset, 2000 will be used for training. Since we can't feed all 2000 tuples of dataset all at once into the model in one iteration (due to RAM limitations), we need to perform batching.

PyTorch provides Dataset and DataLoader classes for performing batching. Dataset class provides an interface between the data source and data loader.

The first thing I did was to create a NewsData class which inherits Dataset. It contains a __get_item__ method which returns encoded x and y of ith tuple.

TRAINING

Next, I created an instance of class DataLoader, specifying Dataset (NewsData in our case) and batch size in its constructor.

This data loader would return random batches of size n whenever called.

Now we need to define loss function and optimiser (optimiser is used for performing gradient descent and updating weights).

I'm using CrossEntropyLoss and Adam Optimiser.

For 150 iterations,

DataLoader will load random batch

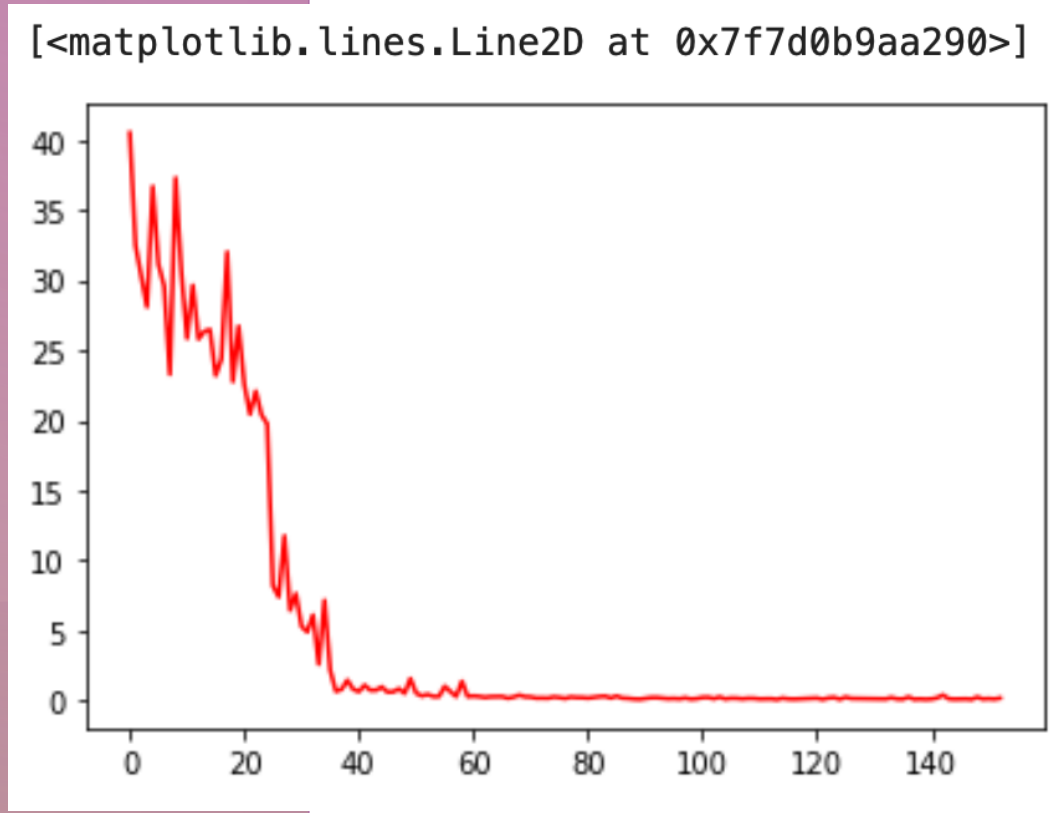
The tuples of this batch would be fed into the model

the model's outputs will be compared with real summaries, loss will be calculated

gradient descent would be performed using optimizer, to update the weights

EVALUATION

THE LOSS DECLINED AS FOLLOWS DURING 150 EPOCHS:



ACCURACY

Now for testing accuracy, traditional accuracy metrics won't be very useful because the words used in generated summaries after bound to differ from actual summaries in testing dataset, even though both might be equally valid.

Hence I instead inferred my evaluation based on how the summaries improved over the epochs.

generated_summary

the the the Army the the of the the army the the army the the the the the army the the the the the army the the the the the the the the the the the the the the

actual_summary

"They are very much not for the good and will destroy Scotland's regiments by moulding them into a single super regiment which will lead to severe recruitment problems, a loss of local connections to those regiments and a loss to Scotland of an important part of her heritage and, most importantly, her future - the regiments are the envy of armies around the world."The proposals to either merge or amalgamate the six regiments into a super regiment sparked a political outcry, with Labour backbenchers

AFTER 25 EPOCHS

generated_summary

The government and the Army Board have spent the past four months attempting to trick serving soldiers and the public into th
inking their plans for restructuring regiments on Monday.The government and the Army Board have spent the past four months at
tempting to trick serving soldiers and the public into thinking their plans for restructuring regiments on Monday.The governm
ent and the Army Board have spent the past four months attempting to trick serving soldiers and the public into thinking thei
r plans for restructuring regiments on Monday.The government and the Army Board have

actual_summary

"They are very much not for the good and will destroy Scotland's regiments by moulding them into a single super regiment whic
h will lead to severe recruitment problems, a loss of local connections to those regiments and a loss to Scotland of an impor
tant part of her heritage and, most importantly, her future - the regiments are the envy of armies around the world."The prop
osals to either merge or amalgamate the six regiments into a super regiment sparked a political outcry, with Labour backbench
ers

AFTER 75 EPOCHS

generated_summary

Under their vision, it would be one of five in the new super regiment.The proposals to either merge or amalgamate the six regiments into a super regiment sparked a political outcry, with Labour backbenchers and opposition politicians opposing the plan.The proposals have faced stiff opposition from campaigners and politicians alike.The government and Army Board have spent the past four months attempting to trick serving soldiers and the public into thinking their planned changes for the Scottish regiments are for the good of the Army and for that

actual_summary

"They are very much not for the good and will destroy Scotland's regiments by moulding them into a single super regiment which will lead to severe recruitment problems, a loss of local connections to those regiments and a loss to Scotland of an important part of her heritage and, most importantly, her future - the regiments are the envy of armies around the world."The proposals to either merge or amalgamate the six regiments into a super regiment sparked a political outcry, with Labour backbenchers

AFTER 150 EPOCHS

OUTCOME

summaries

Search

ONGC Q4 Results | Net profit jumps 31% on high oil, gas prices - Moneycontrol

ONGC reported a 31.5 percent jump in its March quarter net profit as it got the best ever price for crude oil it produces and sells . For the full fiscal (April 2021 to March 2022), ONGC posted a record

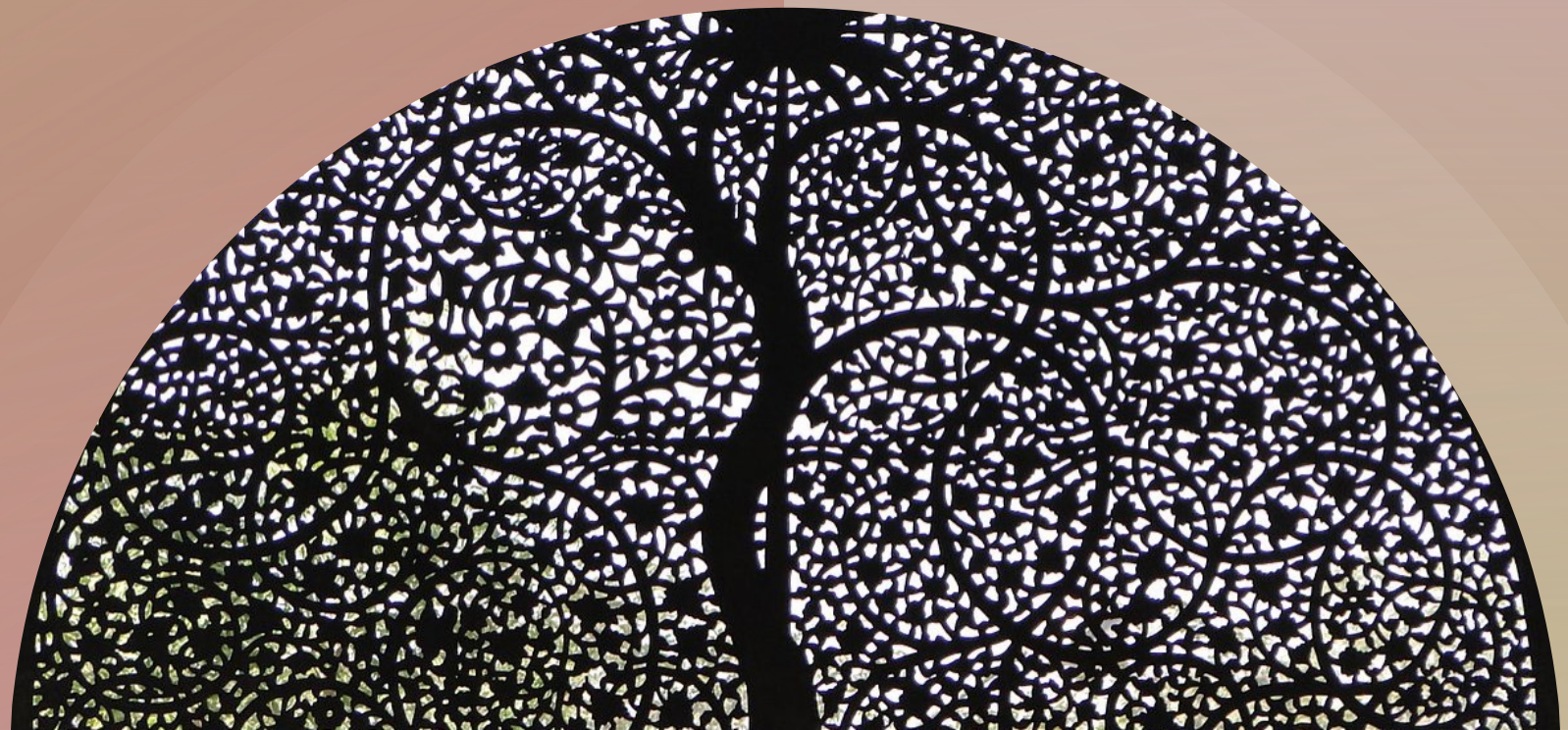
EV troubles, costly fuel fire up sales of CNG cars in India - Economic Times

Domestic sales of CNG personal passenger vehicles in April more than doubled to 27,000 units from a year earlier . Maruti Suzuki expects CNG cars to account for up to 20% of its total sales over the next three to five years .

GDP grows 8.7% in FY22:

Summaries Keywords Recommends

KEYWORDS EXTACTION



INTRODUCTION

- Extract top 5 keywords from the summaries that we have obtained (not 5 keywords from each summary, but 5 from all summaries in total).
- Display those keywords in the Keywords tab of the app.
- Let the users get News articles(summaries) which contain those keywords.

SYSTEM DESIGN OVERVIEW

Create an Azure Function which would be triggered once a day.

It would load all the summaries that we have stored in Summaries container of CosmosDB, iterate through them and extract all nouns from them using POS tagging.

Construct a graph data structure of all the extracted nouns. Nouns will be represented as nodes. Nouns of the same article summary will be connected with edges.

Use PageRank algorithm to rank the importance of nouns.

Use q parameter of NewsAPI's endpoint to load articles based on keywords

Store those articles in CosmosDB, in a container distinct from the one that we used for main summaries.

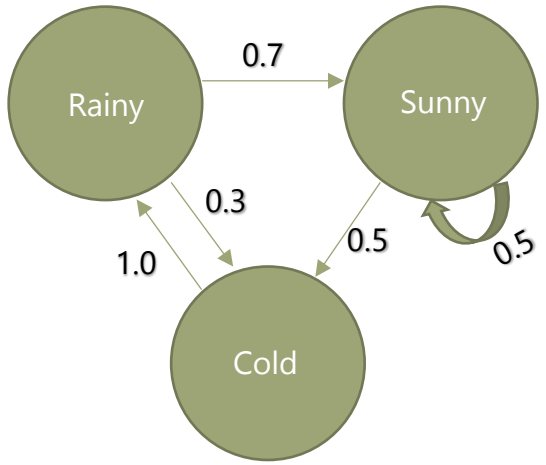
MARKOV CHAIN

MARKOV CHAIN

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

EXAMPLE

- 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

STATE VECTOR

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

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

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

After some iterations, the state vector will stop changing.

$$\pi = P\pi$$
(pi will stop changing)

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

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.

POS TAGGING

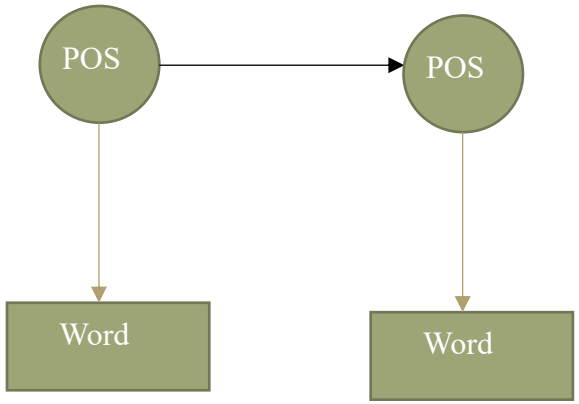
POS TAGGING

Parts of Speech tag means different categories of words, such as noun , preposition , verb , adjective , conjunction , etc

Finding the Part of Speech tag of every word in sentence/para is called POS tagging

POS TAGGING USING HMM

In this model, words are considered as observable states and POS are considered as hidden states.



POS TAGGING USING HMM

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 “Parth” given that the POS tag is noun.

How do we find out transition and emission probabilities? We need to learn them using our 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)

Transition probabilities

- 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).
- Divide each cell by corresponding row total. These would be our transition probabilities.
  Total of each row will be equal to 1.

	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

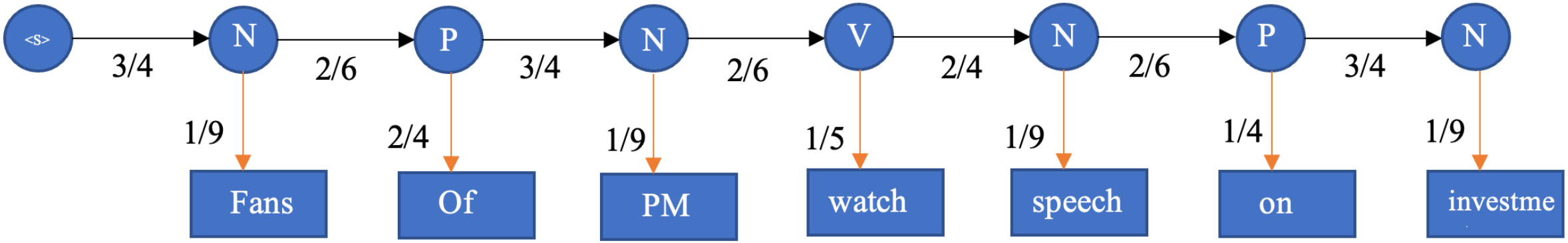
	PRICE	OF	WATCH	SOAR	BECAUSE	INFLATION	PM	SPEECH	ON	TRY	TO	FAN	INVESTMENT
N	2	0	1	0	0	2	1	1	0	0	0	1	1
V	0	0	1	2	0	0	0	0	0	1	0	1	0
P	0	2	0	0	0	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	0	0	0	0	0	0	0

Emission probabilities

- Prepare table of frequencies of each word wrt 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).
- Now divide each cell by corresponding row total. These would be our emission probabilities.

- Now that 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.
  We would calculate join probability for each of the possible sequences.

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



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

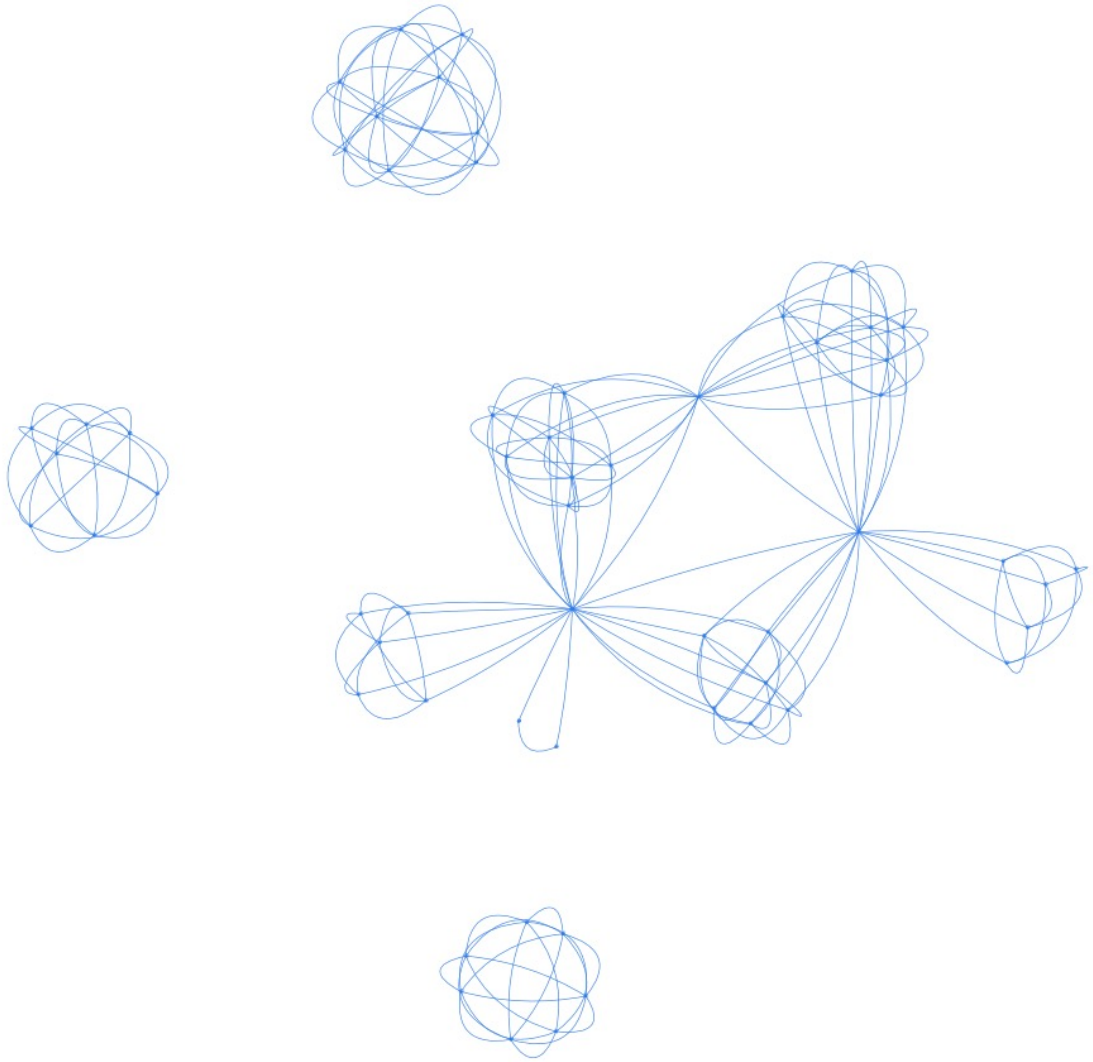
- Similarly we would calculate joint probabilities for all other possible sequences. POS sequence that has the highest join probability will be used for assigning tags.

Once POS tagging is done for all sentences in all article summaries, we would extract nouns for each article summary.

PAGERANK ALGORITHM

GRAPH DATA STRUCTURE

- To find the most important nouns, we first construct a Graph data structure.
- 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 components.
- In the program, to construct the graph from nodelist and edgelist, networkx library is being used.



Visualized using pyvis library

GRAPH AS MARKOV CHAIN

- We can view this graph 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 would be the ones which 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.

PAGERANK

- For the initial state vector, let's consider that the state vector is:
  $\pi = (1/n , 1/n , \dots , 1/n)$
- where ***n*** is the total number of nodes. This means equal probability of being at every state(node).

- **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).

PAGERANK

- 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.
- To fix this, we need to introduce the concepts of teleportation and damping factor.

TELEPORTATION & DAMPING

- We connect 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 the chain.

TELEPORTATION & DAMPING

The transition matrix after introducing teleportation will be:

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

Where P = original transition matrix,

$e^T = (1/n , ..., 1/n)$

$v = (1, 1, ..., 1)^T$

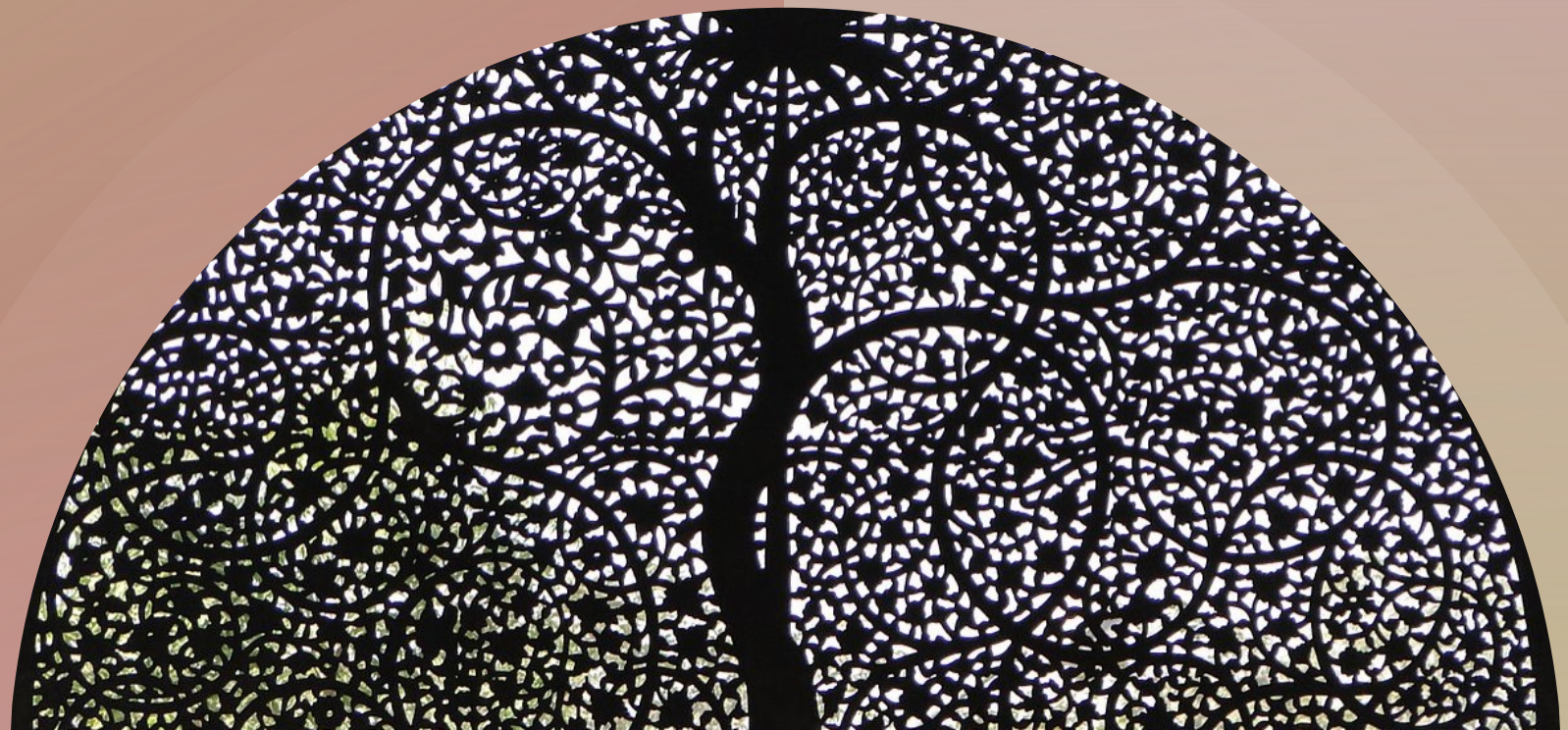
b is the *damping factor*. It is usually set to 0.85.

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.

OUTCOME


keywords
Search
india
april
rs
fed
margins
Summaries
Keywords
Recommends

RECOMMENDATION



SYSTEM DESIGN OVERVIEW

Each document in the CosmosDB “Summaries” database container, has a Boolean field called “isRead”. On the frontend app, when the user opens an article, the POST endpoint “/markRead{id}” of the backend REST API is called. It would update the isRead field in the document corresponding to the id, in CosmosDB “Summaries” database container, to True.

/markRead{id} would then proceed to calculate the similarity of the article which the user is currently reading, with every other article whose corresponding document in the CosmosDB has the field “isRead” set to False.

Top 3 most similar articles would be added to CosmosDB “Recommendations” database container.

On the frontend app, when the user taps on the “Recommends” tab, the GET endpoint “/getRecommends” of our backend REST API is called.

/getRecommends will fetch the documents stored in Azure Cosmos Database using CosmosDB library, parse them, and return them in JSON format.

Frontend will receive this response, deserialize and format it, and show it in a list view.

TYPES OF RECOMMENDER SYSTEMS

COLLABORATIVE FILTERING

- A user's characteristics is compared with characteristics of other users. The articles read by the users with most similar characteristics, are recommended to the current user.
- User's characteristics may be demographics such as age, nationality, gender, etc.
- Cosine similarity between characteristics vector of current user with characteristics vectors of every other users are calculated, and sorted. The articles read by the users who have maximum cosine similarity, are recommended.

CONTENT BASED FILTERING

- It tries to predict user's preferences based on their past activity or based on the things they have reacted positively to before.
- For example, a news dataset may have fields such as "Rating", "Author", "aboutGDP", etc. (This is just an example. Our dataset has no such fields).
- To recommend an article to the user, we would compare similarities between fields of the tuple of the article that user has read, with fields of tuples of every article that the user hasn't read. The articles which have the most similar fields will be recommended to the user.

TF-IDF

For our project, content based filtering is what I've used. But it isn't based on predefined feature set of things like "Rating", "Author", etc. Instead it is based on TF-IDF's of article.

TF-IDF

Term Frequency - Inverse Document Frequency quantifies and vectorizes the importance of words in a document.

It is broken into 2 steps. We first find out Term Frequencies.

TERM FREQUENCIES

01

Construct a bag of words from our summaries(documents) corpus

02

Bag of words is the list of unique words

03

Tokenization and lemmatization will be done on each sentence of each summary, duplicates will be eliminated, and a list of unique words will be constructed

04

In each summary, find frequency of each word contained in bag of words

EXAMPLE

Let's for example say our dataset has two just articles(documents) -

"The farmers protested. The farmers got success in protest."

"Protest started against Agnipath. Protest got no success"

Bag of words will be - ["Farmer", "Protest", "Got", "Success", "Start", "Against", "Agnipath", "No"]

Next, we'll find frequency of each word in each document.

	Farmer	Protest	Got	Success	Start	Against	Agnipath	No
Doc1	2	2	1	1	0	0	0	0
Doc2	0	2	1	1	1	1	1	1

Each row will be TermFrequency vector for the corresponding document.

INVERSE DOCUMENT FREQUENCIES

For each word x in the bag of words, we will calculate its idf as,

Idf(x, D) = log(len(D) / 1 + count(d -> D, x -> d))

Where D is our corpus, len(D) is number of documents in corpus.

In the denominator,
count(d -> D, x -> d) indicates number of documents which contain word x.

IDF will be calculated for each word in bag, and they all will be stored in “IDF” vector

TF-IDF

- Now that we have both - TermFrequency vectors for each document, as well as IDF vector, we can calculate TF-IDF of each document.
- For that, we take cross product of a document d's term frequency vector with the IDF vector.
- TF-IDF(d) = TF(d) x IDF
- We will obtain TF-IDF's for each d.
- The obtained TF-IDF's of each d, are stored in a matrix called "TF_IDF".

TF-IDF

In the project, I have used *feature_extraction.text.TfidfVectorizer* from sklearn library for obtaining TF_IDF matrix.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

TF_IDF = vectorizer.fit_transform(documents)
```

DIMENSIONALITY REDUCTION

DIMENSIONALITY REDUCTION

- Our TF-IDF's are huge, having the dimension of the number of words in corpus. This is a waste of memory and can also increase the time complexity of algorithms which we will apply in future.
- Hence we need to perform dimensionality reduction on our TF_IDF matrix. It will reduce the number of columns in matrix (which implies reducing number of features of each TF-IDF).
- This can be done using Singular Value Decomposition.

SVD

- Our TF-IDF's are huge, having the dimension of the number of words in corpus. This is a waste of memory and can also increase the time complexity of algorithms which we will apply in future.
- Hence we need to perform dimensionality reduction on our TF_IDF matrix. It will reduce the number of columns in matrix (which implies reducing number of features of each TF-IDF).
- This can be done using Singular Value Decomposition.

SVD

We will find SVD of TF_IDF matrix, hence we will find its U, S and V^T matrices.

To reduce dimensionality, we need to eliminate the rows from S in which singular value is negligible. If we removed i^{th} row from S, we also remove i^{th} column from V and i^{th} column from U.

We multiply our new U, S and V^T, and get our new matrix which would have reduced number of columns.

SVD

In the project, I have used *decomposition.TruncatedSVD* from sklearn library for performing dimensionality reduction using SVD.

```
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix
svd = TruncatedSVD(n_components=120, n_iter=7, random_state=44)
TF_IDF_SPARSE = csr_matrix(TF_IDF)
TF_IDF = svd.fit(TF_IDF_SPARSE)
```

CALCULATING SIMILARITIES BETWEEN DOCUMENTS

CALCULATING SIMILARITIES BETWEEN DOCUMENTS

Now that we have dimensionally reduced TF-IDF's, we can use distance metric to calculate similarities

Let's say current article 's TF-IDF is in i^{th} row of our TF_IDF matrix

We will calculate cosine similarities of the i^{th} of our matrix with every other rows

The documents corresponding to rows with which we obtained the least cosine distance, will be recommended.

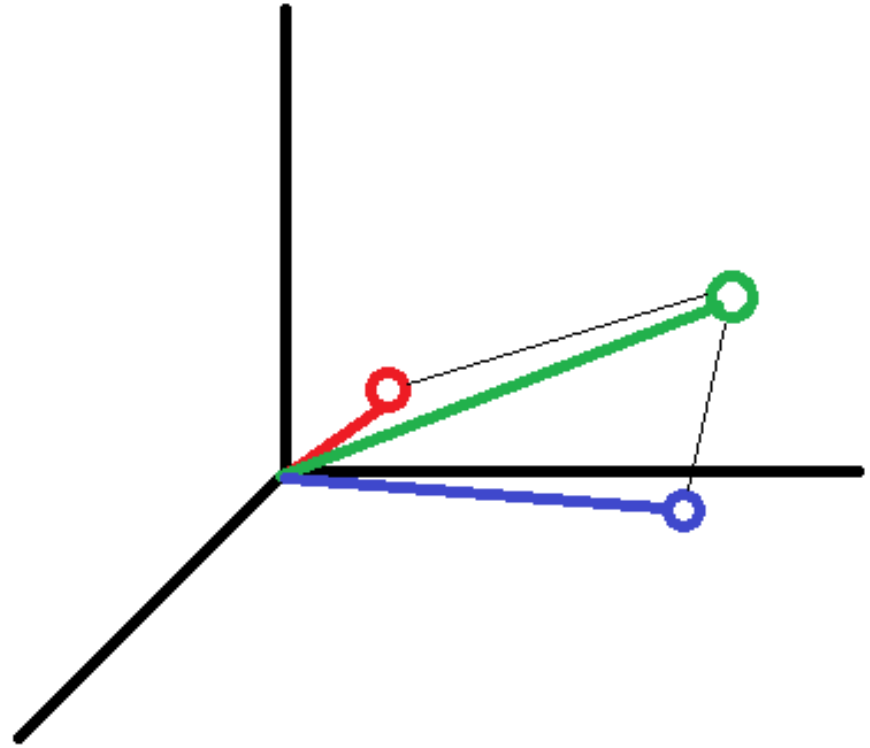
COSINE DISTANCE VS EUCLIDIAN

Let's say document A happens to be much more similar to document B that it is to document C. But merely due to the fact that A's vector has much smaller magnitude than B's vector, it will actually have shorter Euclidian distance with the C's vector than B.

Clearly Euclidian distance isn't giving desirable result. It would be much better to compare angles rather than distances.

That's why we use cosine distance.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$




THANK YOU