## **NLP and Machine Learning - Applications**

# Applications of NLP

So far we have seen basic concepts in NLP such as elements of text, annotations, tagging, handling a corpus of text, and lexical resources.

In this module, we will be using the concepts learnt so far to build the following sample applications:

1. SMS spam classification - Classify if an SMS is spam or ham based on the content in it
2. Sentiment Analysis of product reviews - Classify if a product review has a positive sentiment or a negative sentiment
3. Indian Railways FAQ chatbot - Automate the response to frequently asked questions on Indian Railways ticket booking
4. Topic modelling - Identify key topics in a document and group document that belong to similar topics together

Before we start developing the applications let us review the sequence of steps that we typically go through while developing NLP applications

The typical NLP pipeline involves a variety of steps as listed below:

# Collecting textual data

* This step involves bringing in relevant data from different sources. Sources can include both external and internal systems and depends on the domain of the problem.

# Cleaning the input text data

* We use various techniques to clean the data, depending on the domain of the problem. Regular expressions are extensively used at this stage in identifying anomalies in the data and transforming them as required

# Normalizing the data

* Normalization of text data includes techniques such as "removal of stop words", "converting all text to a standard case", "stemming/lemmatization", etc.

# Feature engineering

* Feature engineering textual data typically involves converting the cleaned and normalized data into vector representations. These vectors can then be used by machine learning algorithms to build models

# Apply machine learning techniques to build models

* Depending on the problem at hand, we can apply different machine learning algorithms. **For example, when we are build a spam classifier/Sentiment Analyzer, we would use popular classification algorithms such as Naive Bayes, Decision Trees, etc. If we are building a system that auto-completes user input, then we may use hidden markov models.**

# Validating the model built

* After building the model, we check it against test data to analyse how the model performs on previously unseen data. If the model performance is satisfactory, we can be more confident in deploying it

# Deploying the model and making prediction on new data.

* The model is now used on real-time data to make predictions/provide responses. For example, when a spam classifier is deployed, it reads messages in real-time and classifies which of those are spam and which are not, and take necessary action.

SMS Spam Classifier using Decision Trees and Naive Bayes

Let us now use the various NLP techniques that we have learnt to build a SMS Spam classifier.

In order to build the classifier, we have collected historic data which has different SMS messages marked as 'ham' (not-spam) or 'spam'. You can download this data from [here](http://10.177.157.28:3006/web-hosted/lex_auth_01259732374489497646/assets/spam.csv).

To build the model, we shall follow the below steps

# Step 1: Load the data into the environment

1. import numpy as np
2. import pandas as pd
3. *# Loading the data into the environment using pandas*
4. *# Note: Please use appropriate filename and path*
5. sms\_data = pd.read\_csv("spam.csv", encoding='latin-1')
6. *# Review the loaded data*
7. print(sms\_data.head())
8. cols = sms\_data.columns[:2]
9. data = sms\_data[cols]
10. print(data.shape)
11. data = data.rename(columns={"v1":"Value","v2":"Text"})
12. print(data.head())
13. print(data.Value.value\_counts())

# Step 2: Feature Engineering

Although this step is titled feature engineering, this includes necessary data cleaning and normalizing operations

1. from string import punctuation
2. import re
3. import nltk
4. from nltk import word\_tokenize
5. punctuation = list(punctuation)
6. *# Creating a new feature called Punctuations.*
7. *# This feature counts the number of punctuation characters in the sms message*
8. data["Punctuations"] = data["Text"].apply(lambda x: len(re.findall(r"[^\w+&&^\s]",x)))
9. *# Creating a new feature called Phonenumbers.*
10. *# This feature indicates if the sms text contains a phonenumber or not*
11. data["Phonenumbers"] = data["Text"].apply(lambda x: len(re.findall(r"[0-9]{10}",x)))
12. *# Creating a new feature called Links.*
13. *# This feature indicates if the sms text contains a URL or not*
14. is\_link = lambda x: 1 if re.search(r"https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+",x)!=None else 0
15. data["Links"] = data["Text"].apply(is\_link)
16. *# Creating a new feature called Uppercase.*
17. *# This feature indicates how many words in the the sms text are in upper case*
18. count\_upper = lambda x : list(map(str.isupper,x.split())).count(True)
19. upper\_case = lambda y,n : n+1 if y.isupper() else n
20. data["Uppercase"] = data["Text"].apply(count\_upper)
21. *# Identifying and counting how many unusual words are there in the sms text*
22. def find\_unusual\_words(text):
23. text\_vocab\_set = set(w.lower() for w in text if w.isalpha())
24. english\_vocab\_set = set(w.lower() for w in nltk.corpus.words.words())
25. unusual\_set = text\_vocab\_set - english\_vocab\_set
26. return len(sorted(unusual\_set))
27. data["unusualwords"] = data["Text"].apply(lambda x: find\_unusual\_words(word\_tokenize(x)))
28. *# View a few records of the data after creating these features*
29. print(data[14:25])

In the above code snippet, we have created new features by understanding the content of our data. This is a critical exercise to undergo when you are building NLP applications.

The below code snippet converts the text into a TF-IDF matrix. Recall that the TF-IDF matrix is a numeric representation of the text.

1. from sklearn.feature\_extraction.text import TfidfVectorizer
2. tf\_idf= TfidfVectorizer(stop\_words="english",strip\_accents='ascii',max\_features=300)
3. tf\_idf\_matrix = tf\_idf.fit\_transform(data["Text"])

TF-IDF vectorization also does some of the required cleaning and normalization steps such as removing punctuation, removing stop words, removing accents, etc. In the above snippet, we have set the value of max\_features to 300, indicating that the TF-IDF matrix contain only the 300 most common words in the text. Doing this reduces the dimensionality of the TF-IDF vector.

Finally, the below code snippet, combines the TF-IDF matrix and the other features we created earlier into a single data frame

1. data\_extra\_features = pd.concat([data,pd.DataFrame(tf\_idf\_matrix.toarray(),columns=tf\_idf.get\_feature\_names())],axis=1)

# Step 3: Machine Learning

We converted our textual data into a type that can be used by machine learning algorithms, let us go ahead and apply machine learning. Here we aim to train the machine to recognize which messages are 'spam' and which are 'ham'

Recall that before we build a machine learning model, we split the data into train and test sets. The model is learnt on the train set and is validated against the test set.

The below code splits the data into train and test sets

1. from sklearn.model\_selection import train\_test\_split
2. X=data\_extra\_features
3. features = X.columns.drop(["Value","Text","Value\_num"])
4. target = ["Value"]
5. X\_train,X\_test,y\_train,y\_test = train\_test\_split(X[features],X[target])

Since the task here is to distinguish one class of messages from another (spam from ham), we need to use classification algorithms.

In the below code, we are using a DecisionTreeClassifier to learn the model

1. from sklearn.tree import DecisionTreeClassifier
2. from sklearn.metrics import accuracy\_score
3. dt = DecisionTreeClassifier(min\_samples\_split=40)
4. dt.fit(X\_train,y\_train)
5. pred = dt.predict(X\_test)
6. print(accuracy\_score(y\_train, dt.predict(X\_train)))
7. print(accuracy\_score(y\_test, pred))

Upon running the code, you can observe that the model has good train and test accuracy.

In the below code snippets, we are building 2 more classifier models - Naive Bayes Classifier and Maximum Entropy Classifier (Logistic Regression)

1. from sklearn.naive\_bayes import MultinomialNB
2. from sklearn.linear\_model import LogisticRegression
3. from sklearn.metrics import accuracy\_score
4. *# Building a Naive Bayes Model*
5. mnb = MultinomialNB()
6. mnb.fit(X\_train,y\_train)
7. pred\_mnb = mnb.predict(X\_test)
8. print(accuracy\_score(y\_test, pred\_mnb))
9. *# Building a Logistic Regression Model*
10. lr = LogisticRegression()
11. lr.fit(X\_train,y\_train)
12. pred\_lr = lr.predict(X\_test)
13. print(accuracy\_score(y\_test, pred\_lr))

Sentiment analysis

# Let us now look at another application of NLP i.e. Sentiment Analysis .

What this means is analyzing the words in the text to define if the overall emotion of the text is a positive or negative one. This is particularly useful in understanding the polarity of a tweet or usefulness of a review on product.

Let us see how to build a basic sentiment analysis application below. Our aim here is to take a list of reviews of different products and analyse them using nltk.

# Step 1: Load the modules into the environment

We will use SentimentIntensityAnalyzer in sentiment.vader a sub module of nltk.

1. *# download vader\_lexicon using nltk.download()*
2. from nltk.sentiment.vader import SentimentIntensityAnalyzer
3. import pandas as pd
4. import numpy as np

# Step 2: Read the csv file and select the desired column

1. fileName = "esteebrandsdata.csv"
2. column = "TextReview"
3. Data = pd.read\_csv(fileName,encoding="Latin-1")
4. Data = Data.replace(np.nan,' ',regex=True)
5. sentences = list(Data[column])

# Step 3: Calculate polarity scores using polarity\_scores method

1. sid = SentimentIntensityAnalyzer()
2. sentiments = []
3. for sentence in sentences:
4. ss = sid.polarity\_scores(sentence)
5. sentiments.append(ss)

# Chatbot

Chat bots, due to the recent advancements in AI, have a huge demand now a days in helping businesses reach and help their customers understand and use their products better.

Let us consider an application in this aspect the **Indian Railways FAQ chat bot**.

The data set below is picked from the most frequently asked questions in Indian Railways Passenger Reservation Enquiry system. The system can be automated using a chat bot for more efficient response.

Our aim here is to create a bot that will take in different kinds of questions from the user and effectively choose a response from its predefined list of responses. Let us begin by importing the necessary modules.

# Step 1: Load the modules into the environment

1. import nltk
2. from nltk.chat.util import Chat, reflections
3. from sys import version\_info
4. from string import punctuation
5. from nltk.corpus import stopwords
6. from nltk.tokenize import word\_tokenize
7. from nltk.stem import PorterStemmer, WordNetLemmatizer

# Step 2: Create appropriate responses for the bots to output

We create pairs of how a question should be and what its response is. Our bot would find the question the input is closest to and output its response related to this question.

1. pairs = [
2. [
3. r"How can I avail internet reservation facility through credit cards?",
4. ['Recently internet reservation facility has started on Indian Railways. The web site http://www.irctc.co.in is operational, wherein you can get the railway reservation done through Credit Cards.For more on Reservation through credit cards click here  Internet Reservation',]
5. ],
6. [
7. r'Why are PNR and reservation availability queries not available after certain timings at night?',
8. ['The online PNR and seat availability queries are fetched from the computerized reservation applications. These online reservation applications are shut down daily around 2330 hrs to 0030 hrs IST. Due to the dynamic changes taking place in the PNR status updation and the availability positions, these two types of queries have to be fetched from the online reservation applications, hence the non- availability of them after certain timings. The sheer size of these databases does not allow them to be copied over network lines.Please note that the web site is functional 24 hrs. a day and other queries (trains between any two stations, fare queries, etc.) are functional throughout the day.',]
9. ],
10. [
11. r'How can I avail the enquiries, through SMS on mobile phones?',
12. ['Now all the enquiries offered on the web site www.indianrail.gov.in are available on your mobile phone through SMS facility. For more information on the mobile service providers and the key words to be used on the mobile, please click here, SMS help . Please note that we are giving the backend service only for the SMS queries. For more information and help on key words and SMS facility, kindly contact the mobile service provider according to the table.',]
13. ],
14. [
15. r'Why do sometimes the fonts, colors schemes and java scripts behave differently in some browser or browsers?',
16. ['This web site is best viewed with Microsoft Internet Explorer 6.0 and above. It might not give desired results with other browsers. All the pages, color schemes and scripts have been tested for IE 6.0 and above. ',]
17. ],
18. [
19. r'Where can I get the latest arrival and departure timings of trains, when they get delayed?',
20. ['The latest arrival and departure timings of delayed trains, alongwith diverted routes etc. will be made available shortly on this web site only.',]
21. ],
22. [
23. r'Where can I lodge complaint against any type of grievances in the Trains, Platforms, officials for problems on this web site and give suggestions?',
24. ['The complaint software is presently under development. We try our best to forward your grievances to the concerned department. However please note that this is not always possible. Please note that all your complaints and suggestions for the improvement of the web site http://www.indianrail.gov.in  can be put on the Feedback & suggestions page. Please note that, in case of any problems, give the query type (hyper link), the inputs which you gave, and the exact error message generated by this web site. All this will help us in solving the problems quickly. In the absence of such inputs, we cannot solve the problems.',]
25. ],
26. ]

Next we will look some preprocessing to feed it to our bot.

# Step 3: Some necessary preprocessing

This step is needed to clean the input - to remove the punctuation, stop words and perform lematization on our resultant data. Next we extract the vocabulary of the sentence using the below function unique().

1. def unique(list1):
2. *# intilize a null list*
3. unique\_list = []
4. *# traverse for all elements*
5. for x in list1:
6. *# check if exists in unique\_list or not*
7. if x not in unique\_list:
8. unique\_list.append(x)
9. return(unique\_list)
10. lemmatiser = WordNetLemmatizer()
11. def preprocessing (sent) :
12. rem\_words = ['get', 'avail', 'who' , 'where', 'how' , 'what', 'why' , 'when', 'I', 'can']
13. *##print(sent)*
14. *# remove punctuation*
15. *# convert to lower*
16. for p in list(punctuation):
17. sent=sent.replace(p,'')
18. sent=sent.lower().split()
19. *#remove stop words*
20. stop\_words = set(stopwords.words('english'))
21. sent = [i for i in sent if not i in stop\_words]
22. sent = [i for i in sent if not i in rem\_words]
23. *# lemmitise*
24. *#[item.upper() for item in mylis]*
25. sent = [lemmatiser.lemmatize(item, pos="v") for item  in sent ]
26. return(unique(sent))

Now that our input looks good after the preprocessing. Let us create the bot and feed the input to get our responses.

# Step 4: Create the bot

In this step the bot accepts input and sends it to our preprocessing methods. We also send our pairs list to the preprocessing methods and get the vocabulary. We then match our input with the list of pairs obtained and check which pair would be the closest to what the input can be.

1. def tellme\_bot():
2. while(1):
3. response = input("Tell Me. [q to quit]>")
4. if response == 'q':
5. break
6. i=0
7. chosen = len(pairs)
8. matches = 0
9. list\_response=preprocessing(response)
10. while (i<len(pairs)):
11. loc\_matches = 0
12. x=pairs[i][0] + "  ".join(pairs[i][1])
13. list\_pair=preprocessing(x)
14. for word in list\_pair:
15. if word in list\_response:
16. loc\_matches=loc\_matches+1
17. if ( loc\_matches > matches ):
18. chosen = i
19. matches = loc\_matches
20. i = i + 1
21. if ( chosen <len(pairs) ) :
22. ans=pairs[chosen][1]
23. print(ans[0] )
24. else :
25. print("Unable to answer this question" )
26. break

Let us now call our bot with input "CAN I RESERVE RAILWAYS BOOKING" and see its response.

1. tellme\_bot()
2. *#prints Tell Me. [q to quit]>CAN I RESERVE RAILWAYS BOOKING*
3. Recently internet reservation facility has started on Indian Railways. The web site http:*//www.irctc.co.in*
4. is operational, wherein you can get the railway reservation done through Credit Cards.For more on Reservation
5. through credit cards click here Internet Reservation

If the bot doesn't understand the question it will respond appropriately as shown below.

1. tellme\_bot()
2. Tell Me. [q to quit]>How do I buy a laptop
3. Unable to answer this question

# Topic Modelling

Topic modelling is an application of NLP. It is a type of statistical modelling to find the topics and its frequency in a particular document.

LDA(Latent Dirichlet allocation) is a type of Topic modelling which we will be using in understanding how Topic modelling works.

The below data set is obtained from the Atkins research which explains why sugar is bad for health. It explains how sugar causes mood swings or how it induces cravings in the body which could give a false sense of hunger. We try to use the below sentences to effectively find out which sample belongs to which topic and try to label each sentence to that topic.

Let us follow similar steps followed in previous applications.

# Step 1: Load necessary modules into our environment

1. from nltk.corpus import stopwords
2. from nltk.stem.wordnet import WordNetLemmatizer
3. import string
4. import gensim
5. from gensim import corpora

# Step 2: Getting data

1. docs1="Sugar causes blood glucose to spike and plummet. Unstable blood sugar often leads to mood swings, fatigue, headaches and cravings for more sugar. Cravings set the stage for a cycle of addiction in which every new hit of sugar makes you feel better temporarily but, a few hours later, results in more cravings and hunger. On the flip side, those who avoid sugar often report having little or no cravings for sugary things and feeling emotionally balanced and energized."
2. docs2="Sugar increases the risk of obesity, diabetes and heart disease. Large-scale studies have shown that the more high-glycemic foods (those that quickly affect blood sugar), including foods containing sugar, a person consumes, the higher his risk for becoming obese and for developing diabetes and heart disease1. Emerging research is also suggesting connections between high-glycemic diets and many different forms of cancer."
3. docs3="Sugar interferes with immune function. Research on human subjects is scant, but animal studies have shown that sugar suppresses immune response5. More research is needed to understand the exact mechanisms; however, we do know that bacteria and yeast feed on sugar and that, when these organisms get out of balance in the body, infections and illness are more likely."
4. docs4="A high-sugar diet often results in chromium deficiency. Its sort of a catch-22. If you consume a lot of sugar and other refined carbohydrates, you probably dont get enough of the trace mineral chromium, and one of chromiums main functions is to help regulate blood sugar. Scientists estimate that 90 percent of Americans dont get enough chromium. Chromium is found in a variety of animal foods, seafood and plant foods. Refining starches and other carbohydrates rob these foods of their chromium supplies."
5. docs5="Sugar accelerates aging. It even contributes to that telltale sign of aging: sagging skin. Some of the sugar you consume, after hitting your bloodstream, ends up attaching itself to proteins, in a process called glycation. These new molecular structures contribute to the loss of elasticity found in aging body tissues, from your skin to your organs and arteries7. The more sugar circulating in your blood, the faster this damage takes hold."
6. docs6="Sugar causes tooth decay. With all the other life-threatening effects of sugar, we sometimes forget the most basic damage it does. When it sits on your teeth, it creates decay more efficiently than any other food substance8. For a strong visual reminder, next time the Tooth Fairy visits, try the old tooth-in-a-glass-of-Coke experiment—the results will surely convince you that sugar isnt good for your pearly whites."
7. docs7="Sugar can cause gum disease, which can lead to heart disease. Increasing evidence shows that chronic infections, such as those that result from periodontal problems, play a role in the development of coronary artery disease9. The most popular theory is that the connection is related to widespread effects from the bodys inflammatory response to infection."
8. docs7="Sugar affects behavior and cognition in children. Though it has been confirmed by millions of parents, most researchers have not been able to show the effect of sugar on childrens behavior. A possible problem with the research is that most of it compared the effects of a sugar-sweetened drink to one containing an artificial sweetener10. It may be that kids react to both real sugar and sugar substitutes, therefore showing no differences in behavior. What about kids ability to learn? Between 1979 and 1983, 803 New York City public schools reduced the amount of sucrose (table sugar) and eliminated artificial colors, flavors and two preservatives from school lunches and breakfasts. The diet policy changes were followed by a 15.7 percent increase in a national academic ranking (previously, the greatest improvement ever seen had been 1.7 percent)."
9. docs8="Sugar increases stress. When were under stress, our stress hormone levels rise; these chemicals are the bodys fight-or-flight emergency crew, sent out to prepare the body for an attack or an escape. These chemicals are also called into action when blood sugar is low. For example, after a blood-sugar spike (say, from eating a piece of birthday cake), theres a compensatory dive, which causes the body to release stress hormones such as adrenaline, epinephrine and cortisol. One of the main things these hormones do is raise blood sugar, providing the body with a quick energy boost. The problem is, these helpful hormones can make us feel anxious, irritable and shaky."
10. docs9="Sugar takes the place of important nutrients. According to USDA data, people who consume the most sugar have the lowest intakes of essential nutrients––especially vitamin A, vitamin C, folate, vitamin B-12, calcium, phosphorous, magnesium and iron. Ironically, those who consume the most sugar are children and teenagers, the individuals who need these nutrients most12."
11. docs10="Slashing Sugar. Now that you know the negative impacts refined sugar can have on your body and mind, youll want to be more careful about the foods you choose. And the first step is getting educated about where sugar lurks—believe it or not, a food neednt even taste all that sweet for it to be loaded with sugar. When it comes to convenience and packaged foods, let the ingredients label be your guide, and be aware that just because something boasts that it is low in carbs or a diet food, doesnt mean its free of sugar. Atkins products never contain added sugar."
12. *# compile documents*
13. doc\_complete=[docs1,docs2,docs3, docs4,docs5,docs6,docs7,docs8,docs9,docs10]

# Step 3: Some necessary preprocessing

1. stop\_set = set(stopwords.words('english'))
2. exclude\_set = set(string.punctuation)
3. lemmatize = WordNetLemmatizer()
4. def clean\_doc(doc):
5. stop\_free = " ".join([i for i in doc.lower().split() if i not in stop\_set])
6. punc\_free = ''.join(i for i in stop\_free if i not in exclude\_set)
7. normalized = " ".join(lemmatize.lemmatize(w) for w in punc\_free.split())
8. return normalized
9. cleaned = [clean\_doc(doc).split() for doc in doc\_complete]

# Step 4: Create LDA model using gensim

1. *# Every unique term is assigned an index in our term document matrix.*
2. dictionary = corpora.Dictionary(cleaned)
3. *# Converting list of documents (corpus) into Document Term Matrix using dictionary prepared above.*
4. doc\_term\_matrix = [dictionary.doc2bow(doc) for doc in cleaned]
5. *# Creating an LDA object*
6. Lda = gensim.models.ldamodel.LdaModel
7. *# Running and Training LDA model on the document term matrix.*
8. ldamodel = Lda(doc\_term\_matrix, num\_topics=5, id2word = dictionary, passes=300)
9. *#Result*
10. topics = ldamodel.print\_topics(num\_topics=5, num\_words=5)
11. for  i in topics :
12. print (i)
13. *#prints*
14. (0, '0.055\*"sugar" + 0.024\*"body" + 0.020\*"blood" + 0.016\*"stress" + 0.016\*"hormone"')
15. (1, '0.029\*"sugar" + 0.020\*"decay" + 0.020\*"tooth" + 0.011\*"result" + 0.011\*"damage"')
16. (2, '0.035\*"sugar" + 0.021\*"behavior" + 0.015\*"percent" + 0.015\*"effect" + 0.015\*"school"')
17. (3, '0.042\*"sugar" + 0.022\*"vitamin" + 0.015\*"consume" + 0.015\*"highglycemic" + 0.015\*"diabetes"')
18. (4, '0.047\*"sugar" + 0.041\*"food" + 0.035\*"chromium" + 0.013\*"get" + 0.013\*"diet"')

# Entity Relationship Extraction

Let us now see how we can extract relationships among different entities present in text. For this demo, you can download the data [here](http://10.177.157.28:3006/web-hosted/lex_auth_0125952578929295361508/assets/A.P.J.AbdulKalam.txt).

# Step 1: Loading the libraries and data into the environment

The below code loads the data into the environment

1. import nltk
2. import pandas as pd
3. text = pd.read\_table('A.P.J.Abdul Kalam.txt',header=None,encoding='utf-8')
4. text = text[0][0]
5. print(text) *#Prints the first paragraph of the article*

# Step 2: Pre-processing the data

In the below code, we are tokenizing and POS tagging all the sentences in the data

1. words = nltk.word\_tokenize(text)
2. tagged\_words = nltk.pos\_tag(words)

# Step 3: Annotating the data and finding entities of interest

In the below code, we are annotating all names and dates that are available in the data.

1. chunkGram = r"""Date: {<CD>?<NNP><CD>}
2. Name: {<NNP>+}"""
3. chunkParser = nltk.RegexpParser(chunkGram)
4. chunked = chunkParser.parse(tagged\_words)
5. print(chunked) *#Prints the annotated text*

# Step 4: Identifying relationship pairs

When two named entities are separated by verbs, prepositions, etc. we can consider that there is some relationship among them.

The below code identifies pairs of relationships in the annotated data.

1. from nltk.sem import relextract
2. pairs = relextract.tree2semi\_rel(chunked)
3. print(pairs)

# Step 5: Extracting the relationship

In the below code we extract the relationship between the pairs identified earlier. We are particularly looking for relationships that have end entity type as Date.

1. relationships = relextract.semi\_rel2reldict(pairs)
2. extracted\_relationships = []
3. for rel in relationships:
4. if rel['objclass'] == "Date" :
5. extracted\_relationships.append(rel['subjtext']+" "+rel['filler']+" "+rel['objtext'])
6. first\_relationship = re.sub(r'/[A-Z]+',"",extracted\_relationships[0])
7. print(first\_relationship)

The output of the above code snippet is "Avul Pakir Jainulabdeen Abdul Kalam was born on 15 October 1931".

# Conditional Random Fields

Conditional Random Fields (CRF) is a sequence modelling technique which is most commonly used for pattern matching using structured prediction.

Generally in other models we do stemming and lemmatization or get vocabulary which makes our data lose its sequentiality or pattern. Which results in a loss of information or context.  CRF tells us that by having a sequential pattern i.e. having the labels of the nearby data(could be tokens or images) a lot of information is taken into account for predicting the labels on the desired column of data.

CRF uses parts of speech of the tokens created from our text to learn the patterns of how each of these tokens occur in different contexts. It can then predict the labels on new data.

For example, given the below sentences tagged with metadata - account number:

My account number is SB100-abc-200/ACCT In this application our aim is to successfully train our model to identify account number out of the text given as an input. For this we will use the sklearn\_crfsuite and nltk.

# Step 1: Load necessary modules

1. import sklearn\_crfsuite
2. import nltk

In the next two steps, let us create a corpus of training data. Using the corpus, we shall then create our features and labels.

# Step 2: Initialize training data

1. stmt1='My account number is SB100-abc-200'
2. stmt2='Can you please tell the balance for CA499-243-520'
3. stmt3='Why is there a debit on my account CC467-923-624 on 10-09-2018 ?'
4. mydata = []
5. stmtlist = [stmt1, stmt2, stmt3]
6. for stmt in stmtlist:
7. stmtw = nltk.word\_tokenize(stmt)
8. stmtl = ['NAA' for x in stmtw]
9. mydata.append(stmtw )
10. mydata.append(stmtl )
11. mydata

Using the mydata we create a list of pairs which will have the word token and its corresponding label

1. data=[(['My', 'account', 'number', 'is', 'SB100-abc-200'], ['NAA', 'NAA', 'NAA', 'NAA', 'ACCT']),
2. (['Can', 'you', 'please', 'tell', 'the', 'balance', 'for', 'CA499-243-520'],['NAA', 'NAA', 'NAA', 'NAA', 'NAA', 'NAA', 'NAA', 'ACCT']),
3. (['Why',  'is',  'there',  'a',  'debit',  'on',  'my',  'account',  'CC467-923-624',  'on',  '10-09-2018',  '?'],
4. ['NAA',  'NAA',  'NAA',  'NAA',  'NAA',  'NAA',  'NAA',  'NAA',  'ACCT',  'NAA',  'NAA',  'NAA'])]

# Step 3: Create corpus

1. *# Create Corpus*
2. def create\_corpus (data):
3. lcorpus = []
4. for (doc, tags) in data:
5. doc\_tag = []
6. for word, tag in zip(doc,tags):
7. doc\_tag.append((word, tag))
8. lcorpus.append(doc\_tag)
10. return (lcorpus)
11. corpus = create\_corpus(data)
12. print(corpus)

* Can you please tell the balance for CA499-243-520/ACCT
* Why is there a debit on my account CC467-923-624/ACCT on 10-09-2018?

Can the system automatically detect the account number in a new text - "Please tell the balance for my account LA233-273-120?"

# Step 4: Extract features from our training data to feed to our CRF model

Let us now create the features from the training data which is then used to create the model. The features in this example is a dictionary containing the current, previous and the next word for every word in the corpus.

1. def convert\_document\_to\_feature\_functions(document, i):
2. *# Building features for feature functions f(s, i , li-1 , l i)*
3. word = document[i][0]
5. features={
6. 'Currword': word,
7. }
8. *# Features from previous word*
9. if i > 0:
10. prevword = document[i-1][0]
11. features['Prevword'] = prevword
12. else:
13. features['BOS'] = True *# Special "Beginning of Sequence" tag*
15. *# Features from next word*
16. if i < len(document)-1:
17. nextword = document[i+1][0]
18. features['Nextword'] = nextword
19. else:
20. features['EOS'] = True *# Special "End of Sequence" tag*
21. return features
22. def extract\_features(doc):
23. *#print('From extract features: ', doc, ': length ' , len(doc))*
24. features = []
25. for i in range(len(doc)):
26. feat=convert\_document\_to\_feature\_functions(doc,i)
27. features.append(feat)
28. return (features)

Let us now create the feature matrix "X"

1. X = [extract\_features(doc) for doc in corpus]
2. for x in X:
3. for dictx in x :
4. print(dictx)

# Step 5: Extracting labels to feed

1. def sentence\_labels(doc):
2. return [tag for (token,tag) in doc]
3. y = [sentence\_labels(doc) for doc in corpus]
4. print(y)

# Step 6: Let us now train our CRF model with the obtained X and y

We create an instance of the sklearn\_crfsuite.CRF and use it to train our model on the obtained feature set and its corresponding labels.

1. crf = sklearn\_crfsuite.CRF(
2. algorithm='lbfgs',
3. c1=0.1,
4. c2=0.1,
5. max\_iterations=20,
6. all\_possible\_transitions=False,
7. )
8. crf.fit(X, y);

# Step 7: Testing our model with new text

Let us first assign "No\_LABEL" to each of the testing data token. And send it to our predict function

1. TestText = 'Please tell the balance for my account LA233-273-120'
2. test\_token = nltk.word\_tokenize(TestText)
3. test=[(x, 'No\_LABEL') for x in test\_token]

Here we see that our CRF model successfully recognized the account number and allocates "ACCT" to it and "NAA" to all other tokens.

1. tcorpus = create\_corpus(test)
2. X\_test = extract\_features(test)
3. print(TestText)
4. print(crf.predict\_single(X\_test))
5. *#prints*
6. Please tell the balance for my account LA233-273-120
7. ['NAA', 'NAA', 'NAA', 'NAA', 'NAA', 'NAA', 'NAA', 'ACCT']