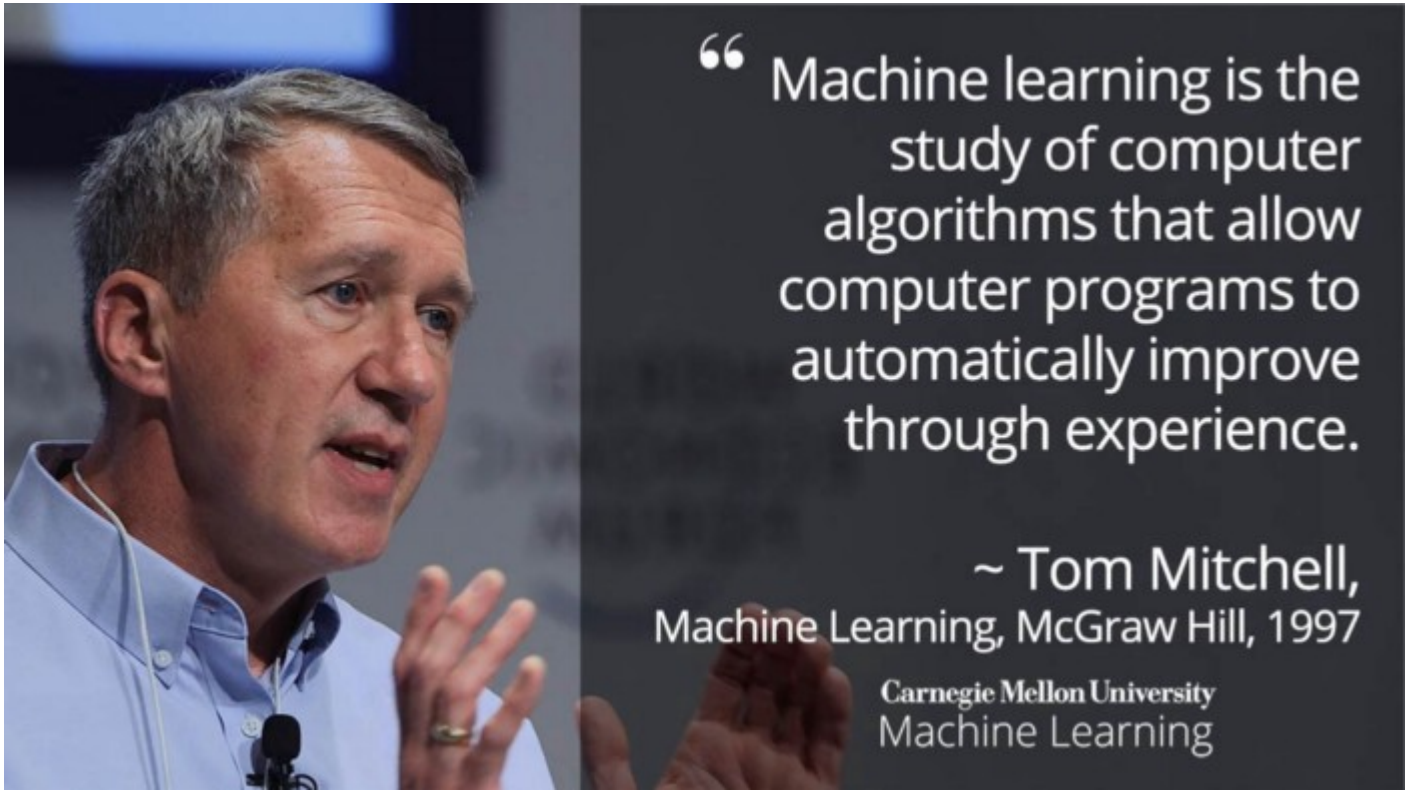
Machine Learning

Learning -> improving with experience.



Computer Scientist and machine learning pioneer Tom M. Mitchell Portrayed | Source: Machine Learning, McGraw Hill, 1997, Tom M. Mitchell https://bit.ly/3J6tuWN

Machine Learning Short Video

https://youtu.be/mmXB636p_E8/

Text Mining Basics

- Wikipedia says:
  - Text mining is "the discovery by computer of new, previously unknown information, by automatically extracting information from different written resources." Written resources can be websites, books, emails, reviews, articles. Text mining, also referred to as text data mining, roughly equivalent to text analytics, is the process of deriving high-quality information from text. High-quality information is typically derived through the devising of patterns and trends through means such as statistical pattern learning. Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluation and interpretation of the output.Typical text mining tasks include text categorization, text clustering, concept/entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and entity relation modeling (i.e., learning relations between named entities).
- Online users participate in different activities and generate a massive amount of unstructured text in differnt forms such as social media posts, articles, website text, blog posts, etc.
- The need to analyze textual data to understand currrent/potenital customer activities, opinion, and feedback is essential for any business. For instance, analyzing custmers feedback expressed in TripAdvisor helps Hotels to improve their service. Exploring tweets helps us to update our knowledge regarding the current news and peoples reaction on events and issues. Online sellers like Amazon can analyze their product reviews to understand the end user experiences.

Text Analytics and Natural Language Processing(NLP)

- NLP assists the computer to understand the human language and extract meaning from it. Its application are popular in various domains such as speech recognition, language translation, classifying documents, information extraction.
- Python provides a opensource package called Natural Language Toolkit (NLTK) which implements a lot of NLP algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition.

In [1]:
```
# install nltk
#!pip install nltk
```

In [2]:
```
#import NLTK
import nltk
```

In [3]:
```
# Lets check what is nltk
help("nltk")
```

```
Help on package nltk:

NAME
    nltk

DESCRIPTION
    The Natural Language Toolkit (NLTK) is an open source Python library
    for Natural Language Processing.  A free online book is available.
    (If you use the library for academic research, please cite the book.)

    Steven Bird, Ewan Klein, and Edward Loper (2009).
    Natural Language Processing with Python.  O'Reilly Media Inc.
    http://nltk.org/book

    @version: 3.6.2

PACKAGE CONTENTS
    app (package)
    book
    ccg (package)
    chat (package)
    chunk (package)
    classify (package)
    cli
    cluster (package)
    collections
    collocations
    compat
    corpus (package)
    data
    decorators
    downloader
    draw (package)
    featstruct
    grammar
    help
    inference (package)
    internals
    jsontags
    lazyimport
    lm (package)
    metrics (package)
    misc (package)
    parse (package)
    probability
```

```
    sem (package)
    sentiment (package)
    stem (package)
    tag (package)
    tbl (package)
    test (package)
    text
    tgrep
    tokenize (package)
    toolbox
    translate (package)
    tree
    treeprettyprinter
    treetransforms
    twitter (package)
    util
    wsd

SUBMODULES
    agreement
    aline
    api
    association
    bleu_score
    bllip
    boxer
    brill
    brill_trainer
    casual
    chart
    chrf_score
    cistem
    confusionmatrix
    corenlp
    crf
    decisiontree
    dependencygraph
    destructive
    discourse
    distance
    drt
    earleychart
    evaluate
    featurechart
    gale_church
```

```
gdfa
gleu_score
glue
hmm
hunpos
ibm1
ibm2
ibm3
ibm4
ibm5
ibm_model
isri
lancaster
legality_principle
lfg
linearlogic
logic
mace
malt
mapping
maxent
megam
meteor_score
mwe
naivebayes
nist_score
nonprojectivedependencyparser
paice
pchart
perceptron
phrase_based
porter
positivenaivebayes
projectivedependencyparser
prover9
punkt
recursivedescent
regexp
relextract
repp
resolution
ribes_score
rslp
rte_classify
scikitlearn
```

```
    scores
    segmentation
    senna
    sequential
    sexpr
    shiftreduce
    simple
    snowball
    sonority_sequencing
    spearman
    stack_decoder
    stanford
    stanford_segmenter
    tableau
    tadm
    textcat
    texttiling
    tnt
    toktok
    transitionparser
    treebank
    viterbi
    weka
    wordnet

FUNCTIONS
    demo()
        # FIXME:  override any accidentally imported demo, see https://github.com/nltk/nltk/issues/2116

    tee(...)
        tee(iterable, n=2) --> tuple of n independent iterators.

DATA
    RUS_PICKLE = 'taggers/averaged_perceptron_tagger_ru/averaged_perceptro...
    SLASH = *slash*
    TYPE = *type*
    __author_email__ = 'stevenbird1@gmail.com'
    __classifiers__ = ['Development Status :: 5 - Production/Stable', 'Int...
    __copyright__ = 'Copyright (C) 2001-2021 NLTK Project.\n\nDistribut......
    __keywords__ = ['NLP', 'CL', 'natural language processing', 'computati...
    __license__ = 'Apache License, Version 2.0'
    __longdescr__ = 'The Natural Language Toolkit (NLTK) is a Python ... p...
    __maintainer__ = 'Steven Bird, Edward Loper, Ewan Klein'
    __maintainer_email__ = 'stevenbird1@gmail.com'
    __url__ = 'http://nltk.org/'
```

```
    app = <LazyModule 'nltk.nltk.app'>
    chat = <LazyModule 'nltk.nltk.chat'>
    corpus = <LazyModule 'nltk.nltk.corpus'>
    infile = <_io.TextIOWrapper name='/home/bidur/.local/lib/...packages/n...
    json_tags = {'!nltk.tag.BrillTagger': <class 'nltk.tag.brill.BrillTagg...
    toolbox = <LazyModule 'nltk.nltk.toolbox'>
    version_file = '/home/bidur/.local/lib/python3.6/site-packages/nltk/VE...
    version_info = sys.version_info(major=3, minor=6, micro=9, releaseleve...

VERSION
    3.6.2

AUTHOR
    Steven Bird, Edward Loper, Ewan Klein

FILE
    /home/bidur/.local/lib/python3.6/site-packages/nltk/__init__.py
```

In [4]:
```
# what does it contain
dir("nltk")
```

Out[4]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
```

```
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmod__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'capitalize',
'casefold',
'center',
'count',
'encode',
'endswith',
'expandtabs',
'find',
'format',
'format_map',
'index',
'isalnum',
'isalpha',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
'join',
'ljust',
'lower',
'lstrip',
'maketrans',
'partition',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
```

```
'rsplit',
'rstrip',
'split',
'splitlines',
'startswith',
'strip',
'swapcase',
'title',
'translate',
'upper',
```

Tokenization:

- breaking down a text paragraph into smaller chunks such as words or sentence.
- Token: single entity, i.e. building blocks for sentence or paragraph.

In [5]:
```
# sentence tokenization
from nltk.tokenize import sent_tokenize
text="""Machine Learning Example Class, WRC Pokhara, Nepal.
WRC is in Pokhara. Pokhara is a popular tourism destination in Nepal.
Nepal is a south asian country. Let's mine the knowledge within."""
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Machine Learning Example Class, WRC Pokhara, Nepal.', 'WRC is in Pokhara.', 'Pokhara is a popular tourism
destination in Nepal.', 'Nepal is a south asian country.', "Let's mine the knowledge within."]
```

In [6]:
```
# word tokenization

from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Machine', 'Learning', 'Example', 'Class', ',', 'WRC', 'Pokhara', ',', 'Nepal', '.', 'WRC', 'is', 'in', 'Po
khara', '.', 'Pokhara', 'is', 'a', 'popular', 'tourism', 'destination', 'in', 'Nepal', '.', 'Nepal', 'is', '
a', 'south', 'asian', 'country', '.', 'Let', "'s", 'mine', 'the', 'knowledge', 'within', '.']
```

Frequency DIstribution:

- counting words in the text.
- Class FreqDist works in a dictionary like manner and generates a key-value pair.
  - Keys: words in the input

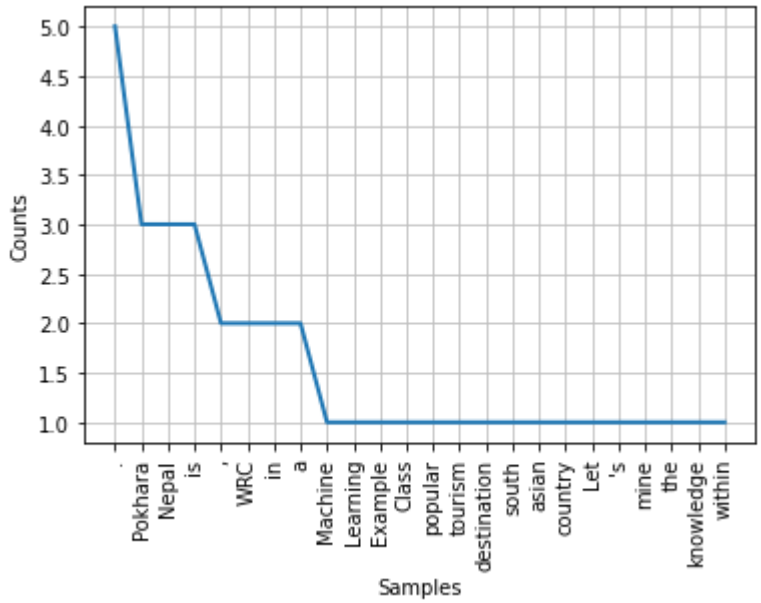
In [7]:
```
from nltk.probability import FreqDist
fdist = FreqDist(tokenized_word)
print(fdist)
print ("Nepal: %d" % fdist['Nepal']) # gives the count of 'Nepal' in the input
print ("Pokhara: " + str(fdist['WRC']) )
```

```
<FreqDist with 24 samples and 38 outcomes>
Nepal: 3
Pokhara: 2
```

In [8]:
```
fdist.most_common(2)
```

Out[8]:
```
[('.', 5), ('Pokhara', 3)]
```

In [9]:
```
# Lets plot the frequency distribution
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
plt.show()
```



Stopwords

- These are the common words in a language such as is, am, are, this, a, an, the, etc.

- normally considered as noise

In [10]:
```
# Lets check the NLTK stopwords
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'the', 'so', 'other', 'hadn', 'but', 'of', 'is', "won't", 'an', 'am', "weren't", 'until', "shan't", 'had',
'ma', 'isn', 'both', 'there', 'under', 'against', 'themselves', 'he', 'when', 'doesn', 'her', 'its', 'just',
'our', 'then', 'been', 'which', 'after', 'itself', 'all', 'most', 'aren', "doesn't", 'where', 'too', 'hasn',
'down', 'yours', "hasn't", 'wasn', 'won', 'whom', 'can', "hadn't", "couldn't", 'below', 'about', "don't", 'f
ew', 'you', 'have', 'again', 'a', 'i', 'this', 's', 'was', "you've", 'as', 'couldn', "you'd", 'any', 'own',
'are', 'for', 'than', 've', 'what', "didn't", "wouldn't", 'should', 'being', 'how', 'such', 'further', 'beca
use', "aren't", 'or', 'll', 'some', "isn't", 'who', 'y', 'only', 'same', "mustn't", 'yourselves', 'nor', "sh
ould've", 'shouldn', 'wouldn', 'your', 'through', 'o', "you'll", "needn't", 'their', 'it', 'very', 'weren',
'those', 'did', "you're", 're', 'me', 'more', 't', "shouldn't", 'them', 'with', 'at', 'd', 'were', 'she', 'n
ot', 'has', 'having', 'himself', 'theirs', 'by', 'mightn', 'before', 'on', 'shan', 'we', 'him', 'ourselves',
'yourself', 'didn', 'while', 'now', 'm', 'ours', 'to', 'haven', 'into', "mightn't", 'myself', 'they', 'ain',
'mustn', 'above', 'does', 'once', 'hers', "wasn't", 'during', 'will', "she's", "that'll", 'if', 'here', 'my
', 'from', 'between', 'and', 'don', 'each', 'needn', 'that', 'be', 'over', 'in', "haven't", 'why', 'out', 'o
ff', 'no', 'his', "it's", 'up', 'doing', 'herself', 'these', 'do'}
```

Since stopwords are considered as noise, we have to remove them

In [11]:
```
filtered_sent=[]
for w in tokenized_word:
    if w not in stop_words:
        filtered_sent.append(w)
print("Tokenized Sentence:",tokenized_word)
print("Filterd Sentence:",filtered_sent)
```

```
Tokenized Sentence: ['Machine', 'Learning', 'Example', 'Class', ',', 'WRC', 'Pokhara', ',', 'Nepal', '.', 'W
RC', 'is', 'in', 'Pokhara', '.', 'Pokhara', 'is', 'a', 'popular', 'tourism', 'destination', 'in', 'Nepal',
'.', 'Nepal', 'is', 'a', 'south', 'asian', 'country', '.', 'Let', "'s", 'mine', 'the', 'knowledge', 'within
', '.']
Filterd Sentence: ['Machine', 'Learning', 'Example', 'Class', ',', 'WRC', 'Pokhara', ',', 'Nepal', '.', 'WRC
', 'Pokhara', '.', 'Pokhara', 'popular', 'tourism', 'destination', 'Nepal', '.', 'Nepal', 'south', 'asian',
'country', '.', 'Let', "'s", 'mine', 'knowledge', 'within', '.']
```

Lexicon

- It is also called as wordbook i.e. the complete set of meaningful units in a language.
- Lexicon normalization considers another type of noise in the text. For example, mining, mines, mined word reduce to a common word "mine". It reduces derivationally related forms of a word to a common root word.

Stemming

- process of linguistic normalization.
- It reduces words to their root word. For example, mining, mines, mined word reduce to a common word "mine"

In [12]:
```
# Lets check the stem words in the input
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()

stemmed_words=[]
for w in filtered_sent:
    stemmed_words.append(ps.stem(w))

print("Filtered Sentence:",filtered_sent)
print("Stemmed Sentence:",stemmed_words)
```

```
Filtered Sentence: ['Machine', 'Learning', 'Example', 'Class', ',', 'WRC', 'Pokhara', ',', 'Nepal', '.', 'WR
C', 'Pokhara', '.', 'Pokhara', 'popular', 'tourism', 'destination', 'Nepal', '.', 'Nepal', 'south', 'asian',
'country', '.', 'Let', "'s", 'mine', 'knowledge', 'within', '.']
Stemmed Sentence: ['machin', 'learn', 'exampl', 'class', ',', 'wrc', 'pokhara', ',', 'nepal', '.', 'wrc', 'p
okhara', '.', 'pokhara', 'popular', 'tourism', 'destin', 'nepal', '.', 'nepal', 'south', 'asian', 'countri',
'.', 'let', "'s", 'mine', 'knowledg', 'within', '.']
```

Lemmatization

- Reduces words to their base word i.e. lemmas.
- It transforms root word with the use of vocabulary and morphological analysis. Lemmatization is usually more sophisticated than stemming. Stemmer works on an individual word without knowledge of the context. For example, The word "better" has "good" as its lemma. This thing will miss by stemming because it requires a dictionary look-up.

In [13]:
```
#Lets compare stemming and Lemmatization. Which is better??

from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()

word = "flying"
print("Lemmatized Word:",lem.lemmatize(word,"v"))
print("Stemmed Word:",stem.stem(word))
```

```
Lemmatized Word: fly
Stemmed Word: fli
```

Part-of-Speech (POS) Tagging

- identifies the grammatical group of a given word such as NOUN, ADJECTIVE, VERB, etc. based on the context.

In [14]:
```
sent = " Nepal experienced a devastating earthquake in 2015."

tokens=nltk.word_tokenize(sent)
print ("tokens: ", tokens)

print ("POS: ", nltk.pos_tag(tokens) )
```

```
tokens:  ['Nepal', 'experienced', 'a', 'devastating', 'earthquake', 'in', '2015', '.']
POS:  [('Nepal', 'NNP'), ('experienced', 'VBD'), ('a', 'DT'), ('devastating', 'VBG'), ('earthquake', 'NN'),
('in', 'IN'), ('2015', 'CD'), ('.', '.')]
```

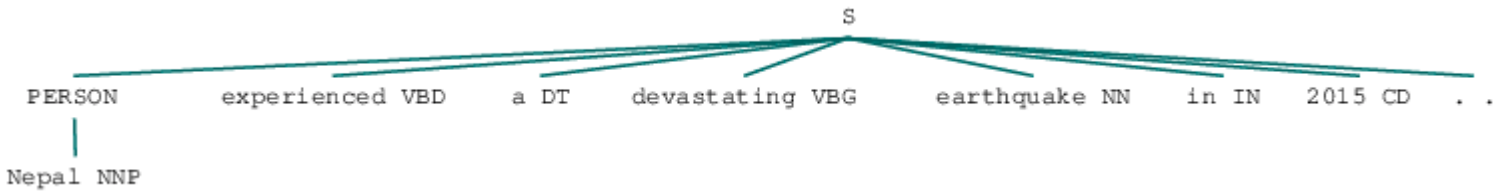
Named Entity Recognisition

- detecting the named entities like location,company, person, monetary value, etc.

In [15]:
```
#nltk.download('maxent_ne_chunker')
```

In [16]:
```
text = " Nepal experienced a devastating earthquake in 2015."
#importing chunk library from nltk
from nltk import ne_chunk
# tokenize and POS Tagging before doing chunk
token = word_tokenize(text)
tags = nltk.pos_tag(token)
chunk = ne_chunk(tags)
chunk
```

Out[16]:



Chunking

- grouping individual pieces of information into a bigger pieces, i.e., grouping of words or tokens into chunks.

In [17]:
```
text = " The white paper is the big black bag"
token = word_tokenize(text)
tags = nltk.pos_tag(token)
reg = "NP: {<DT>?<JJ>*<NN>}"
a = nltk.RegexpParser(reg)
result = a.parse(tags)
print(result)
```

```
(S
  (NP The/DT white/JJ paper/NN)
  is/VBZ
  (NP the/DT big/JJ black/JJ bag/NN))
```

In [ ]:

In [ ]:

In [ ]:

References:

- https://en.wikipedia.org/wiki/Text_mining
- https://expertsystem.com/natural-language-processing-and-text-mining/
- https://www.nltk.org/
- https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk

In [ ]: