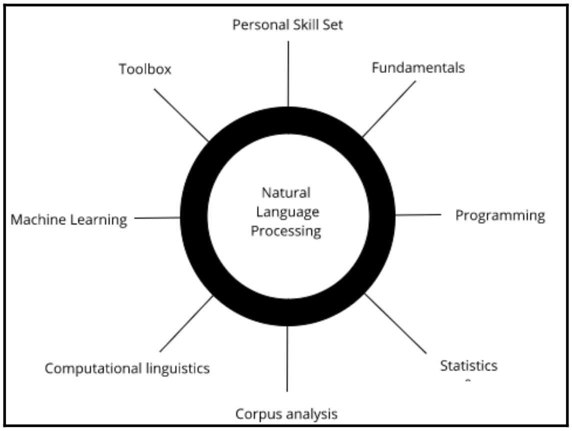
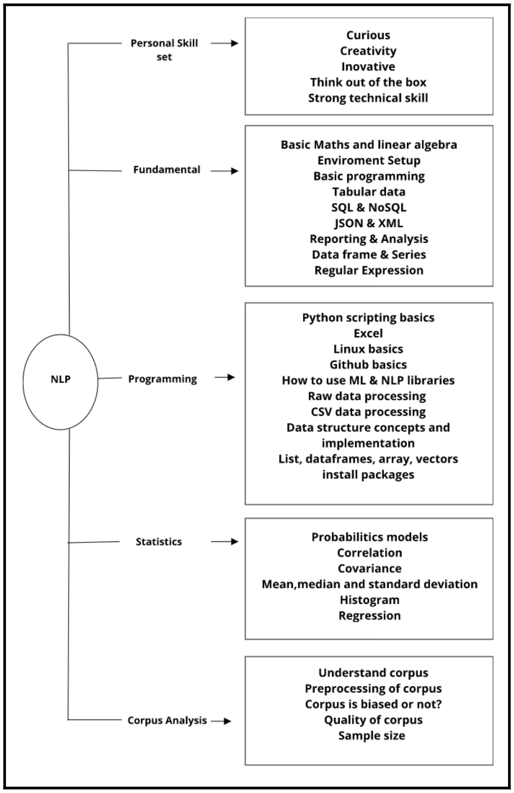
Graphics Bundle

Chapter 1: Introduction





NLP

Personal Skill set

Curious
Creativity
Inovative
Think out of the box
Strong technical skill

Fundamental

Basic Maths and linear algebra
Enviroment Setup
Basic programming
Tabular data
SQL & NoSQL
JSON & XML
Reporting & Analysis
Data frame & Series
Regular Expression

Programming

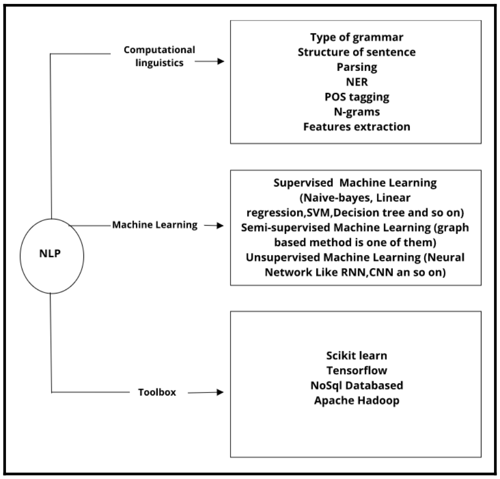
Python scripting basics
Excel
Linux basics
Github basics
How to use ML & NLP libraries
Raw data processing
CSV data processing
Data structure concepts and implementation
List, dataframes, array, vectors
install packages

Statistics

Probabilitics models
Correlation
Covariance
Mean,median and standard deviation
Histogram
Regression

Corpus Analysis

Understand corpus
Preprocessing of corpus
Corpus is biased or not?
Quality of corpus
Sample size



NLP

Computational linguistics

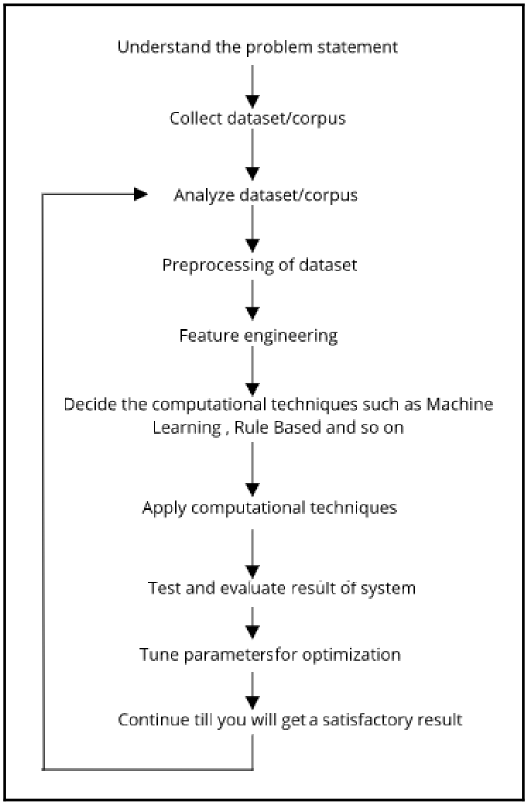
Type of grammar
Structure of sentence
Parsing
NER
POS tagging
N-grams
Features extraction

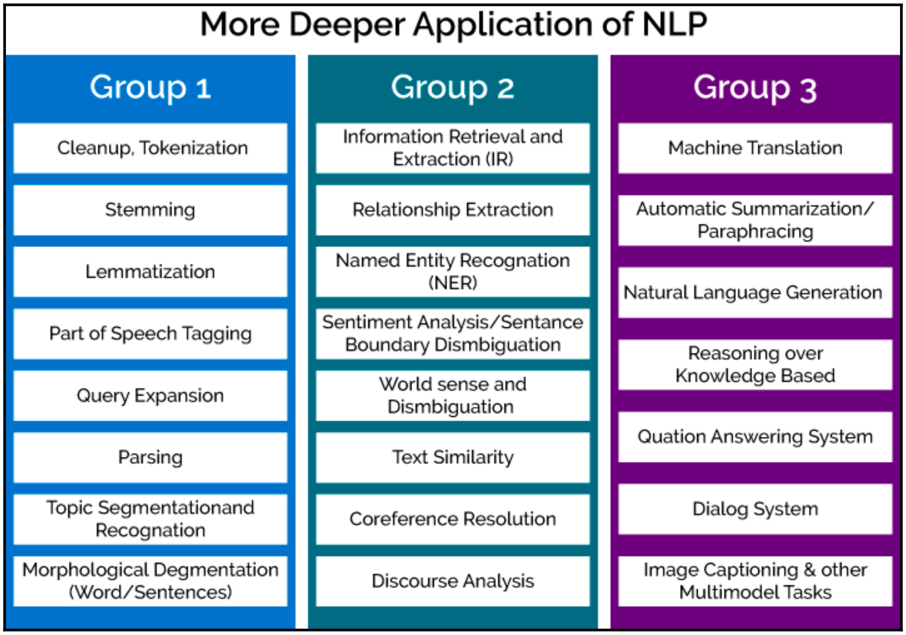
Machine Learning

Supervised Machine Learning (Naive-bayes, Linear regression,SVM,Decision tree and so on)
Semi-supervised Machine Learning (graph based method is one of them)
Unsupervised Machine Learning (Neural Network Like RNN,CNN an so on)

Toolbox

Scikit learn
Tensorflow
NoSql Databased
Apache Hadoop


Understand the problem statement
Collect dataset/corpus
Analyze dataset/corpus
Preprocessing of dataset
Feature engineering
Decide the computational techniques such as Machine Learning , Rule Based and so on
Apply computational techniques
Test and evaluate result of system
Tune parametersfor optimization
Continue till you will get a satisfactory result



More Deeper Application of NLP

Group 1

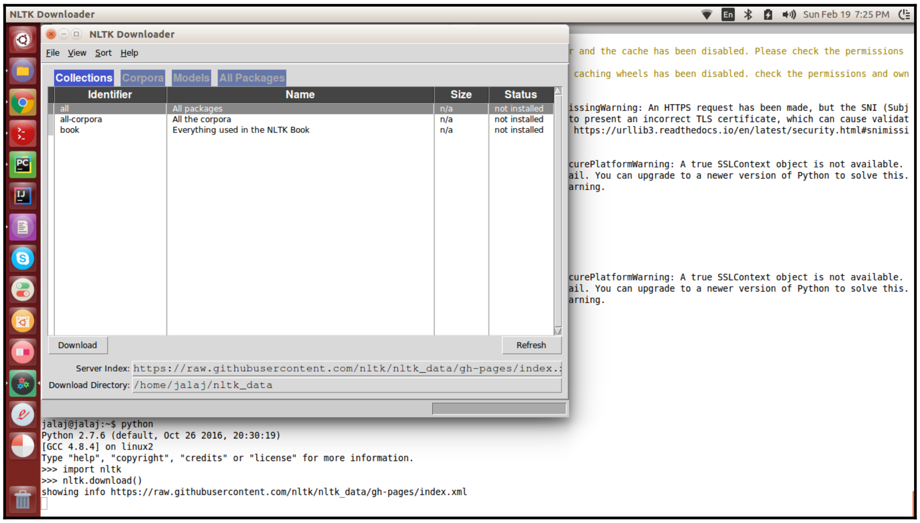
- Cleanup, Tokenization
- Stemming
- Lemmatization
- Part of Speech Tagging
- Query Expansion
- Parsing
- Topic Segmentationand Recognation
- Morphological Degmentation (Word/Sentences)

Group 2

- Information Retrieval and Extraction (IR)
- Relationship Extraction
- Named Entity Recognation (NER)
- Sentiment Analysis/Sentance Boundary Dismbiguation
- World sense and Dismbiguation
- Text Similarity
- Coreference Resolution
- Discourse Analysis

Group 3

- Machine Translation
- Automatic Summarization/Paraphracing
- Natural Language Generation
- Reasoning over Knowledge Based
- Quation Answering System
- Dialog System
- Image Captioning & other Multimodel Tasks



Identifier	Name	Size	Status
all	All packages	n/a	not installed
all-corpora	All the corpora	n/a	not installed
book	Everything used in the NLTK Book	n/a	not installed

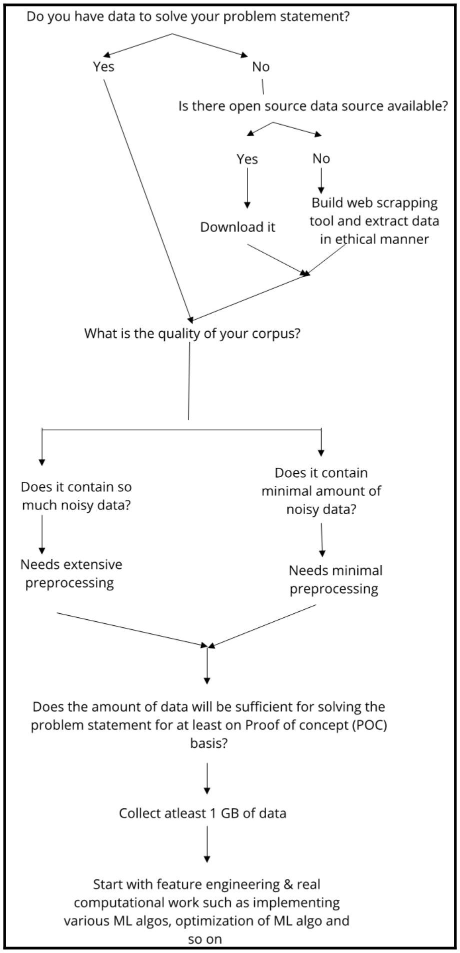
Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.:

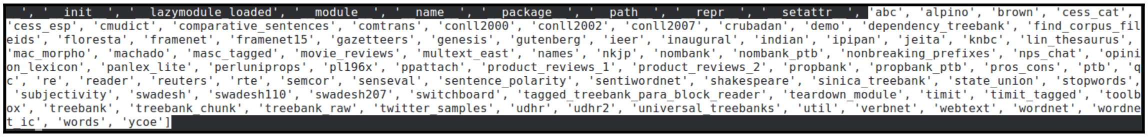
Download Directory: /home/jalaj/nltk_data

```
r and the cache has been disabled. Please check the permissions

 caching wheels has been disabled. check the permissions and own


issingWarning: An HTTPS request has been made, but the SNI (Subj
to present an incorrect TLS certificate, which can cause validat
 https://urllib3.readthedocs.io/en/latest/security.html#snimissi


curePlatformWarning: A true SSLContext object is not available.
ail. You can upgrade to a newer version of Python to solve this.
arning.




curePlatformWarning: A true SSLContext object is not available.
ail. You can upgrade to a newer version of Python to solve this.
arning.
```

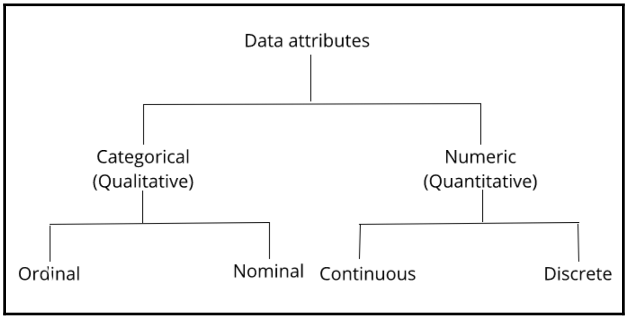
```
jalaj@jalaj:~$ python
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

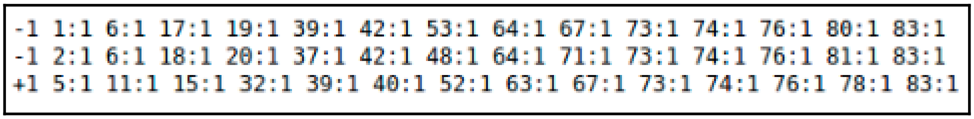
Chapter 2: Practical Understanding of Corpus and Dataset


Do you have data to solve your problem statement?
Yes
No
Is there open source data source available?
Yes
No
Download it
Build web scrapping tool and extract data in ethical manner
What is the quality of your corpus?
Does it contain so much noisy data?
Does it contain minimal amount of noisy data?
Needs extensive preprocessing
Needs minimal preprocessing
Does the amount of data will be sufficient for solving the problem statement for at least on Proof of concept (POC) basis?
Collect atleast 1 GB of data
Start with feature engineering & real computational work such as implementing various ML algos, optimization of ML algo and so on

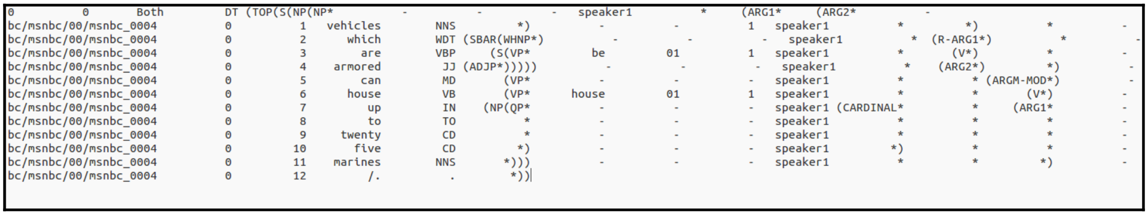


```
__', '__init__', '__lazymodule_loaded', '__module__', '__name__', '__package__', '__path__', '__repr__', '__setattr__', 'abc', 'alpino', 'brown', 'cess_cat',
'cess_esp', 'cmudict', 'comparative_sentences', 'comtrans', 'conll2000', 'conll2002', 'conll2007', 'crubadan', 'demo', 'dependency_treebank', 'find_corpus_fil
eids', 'floresta', 'framenet', 'framenet15', 'gazetteers', 'genesis', 'gutenberg', 'ieer', 'inaugural', 'indian', 'ipipan', 'jeita', 'knbc', 'lin_thesaurus',
'mac_morpho', 'machado', 'masc_tagged', 'movie_reviews', 'multext_east', 'names', 'nkjp', 'nombank', 'nombank_ptb', 'nonbreaking_prefixes', 'nps_chat', 'opini
on_lexicon', 'panlex_lite', 'perluniprops', 'pl196x', 'ppattach', 'product_reviews_1', 'product_reviews_2', 'propbank', 'propbank_ptb', 'pros_cons', 'ptb', 'q
c', 're', 'reader', 'reuters', 'rte', 'semcor', 'senseval', 'sentence_polarity', 'sentiwordnet', 'shakespeare', 'sinica_treebank', 'state_union', 'stopwords',
 'subjectivity', 'swadesh', 'swadesh110', 'swadesh207', 'switchboard', 'tagged_treebank_para_block_reader', 'teardown_module', 'timit', 'timit_tagged', 'toolb
ox', 'treebank', 'treebank_chunk', 'treebank_raw', 'twitter_samples', 'udhr', 'udhr2', 'universal_treebanks', 'util', 'verbnet', 'webtext', 'wordnet', 'wordne
t_ic', 'words', 'ycoe']
```

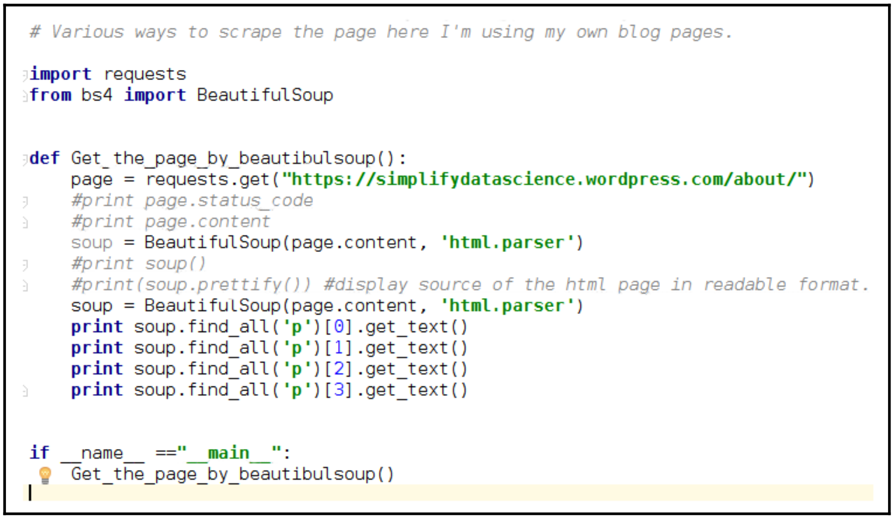




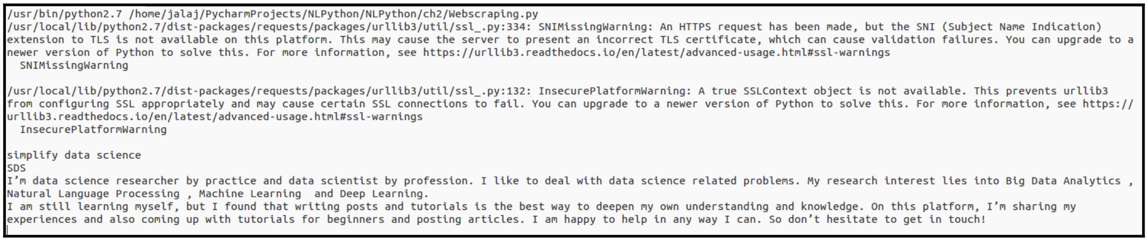
```
-1 1:1 6:1 17:1 19:1 39:1 42:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 20:1 37:1 42:1 48:1 64:1 71:1 73:1 74:1 76:1 81:1 83:1
+1 5:1 11:1 15:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
```



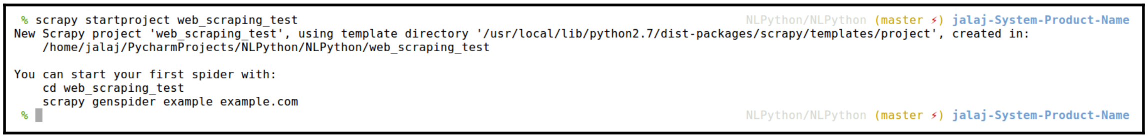
```
0          0       Both         DT (TOP(S(NP(NP*          -          -          -   speaker1          *     (ARG1*     (ARG2*          -
bc/msnbc/00/msnbc_0004          0          1   vehicles        NNS         *)          -          -          1   speaker1          *         *)          *          -
bc/msnbc/00/msnbc_0004          0          2      which        WDT (SBAR(WHNP*)          -          -          -   speaker1          * (R-ARG1*)           *          -
bc/msnbc/00/msnbc_0004          0          3        are        VBP     (S(VP*         be         01          1   speaker1          *       (V*)          *          -
bc/msnbc/00/msnbc_0004          0          4    armored         JJ (ADJP*)))))          -          -          -   speaker1          *   (ARG2*)          *)          -
bc/msnbc/00/msnbc_0004          0          5        can         MD       (VP*          -          -          -   speaker1          *          * (ARGM-MOD*)          -
bc/msnbc/00/msnbc_0004          0          6      house         VB       (VP*      house         01          1   speaker1          *          *       (V*)          -
bc/msnbc/00/msnbc_0004          0          7         up         IN    (NP(QP*          -          -          -   speaker1 (CARDINAL*          *     (ARG1*          -
bc/msnbc/00/msnbc_0004          0          8         to         TO          *          -          -          -   speaker1          *          *          *          -
bc/msnbc/00/msnbc_0004          0          9     twenty         CD          *          -          -          -   speaker1          *          *          *          -
bc/msnbc/00/msnbc_0004          0         10       five         CD         *)          -          -          -   speaker1         *)          *          *          -
bc/msnbc/00/msnbc_0004          0         11    marines        NNS       *)))          -          -          -   speaker1          *          *         *)          -
bc/msnbc/00/msnbc_0004          0         12         /.          .        *))
```



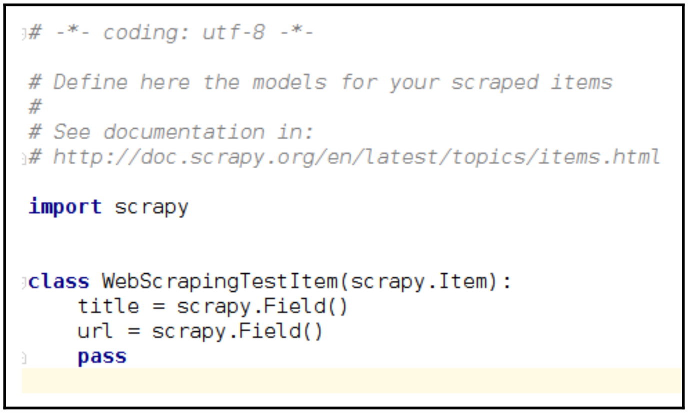
```
# Various ways to scrape the page here I'm using my own blog pages.

import requests
from bs4 import BeautifulSoup


def Get_the_page_by_beautibulsoup():
    page = requests.get("https://simplifydatascience.wordpress.com/about/")
    #print page.status_code
    #print page.content
    soup = BeautifulSoup(page.content, 'html.parser')
    #print soup()
    #print(soup.prettify()) #display source of the html page in readable format.
    soup = BeautifulSoup(page.content, 'html.parser')
    print soup.find_all('p')[0].get_text()
    print soup.find_all('p')[1].get_text()
    print soup.find_all('p')[2].get_text()
    print soup.find_all('p')[3].get_text()


if __name__ =="__main__":
    Get_the_page_by_beautibulsoup()
```



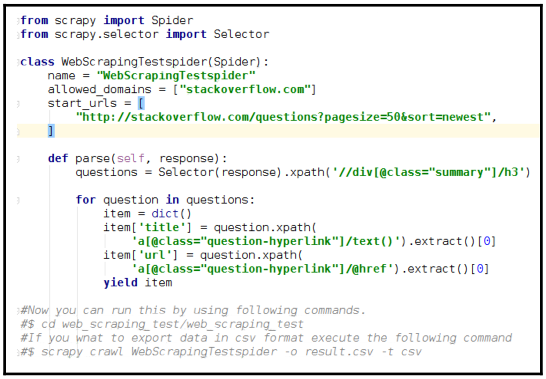
```
/usr/bin/python2.7 /home/jalaj/PycharmProjects/NLPython/NLPython/ch2/Webscraping.py
/usr/local/lib/python2.7/dist-packages/requests/packages/urllib3/util/ssl_.py:334: SNIMissingWarning: An HTTPS request has been made, but the SNI (Subject Name Indication)
extension to TLS is not available on this platform. This may cause the server to present an incorrect TLS certificate, which can cause validation failures. You can upgrade to a
newer version of Python to solve this. For more information, see https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  SNIMissingWarning

/usr/local/lib/python2.7/dist-packages/requests/packages/urllib3/util/ssl_.py:132: InsecurePlatformWarning: A true SSLContext object is not available. This prevents urllib3
from configuring SSL appropriately and may cause certain SSL connections to fail. You can upgrade to a newer version of Python to solve this. For more information, see https://
urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecurePlatformWarning

simplify data science
SDS
I'm data science researcher by practice and data scientist by profession. I like to deal with data science related problems. My research interest lies into Big Data Analytics ,
Natural Language Processing , Machine Learning  and Deep Learning.
I am still learning myself, but I found that writing posts and tutorials is the best way to deepen my own understanding and knowledge. On this platform, I'm sharing my
experiences and also coming up with tutorials for beginners and posting articles. I am happy to help in any way I can. So don't hesitate to get in touch!
```



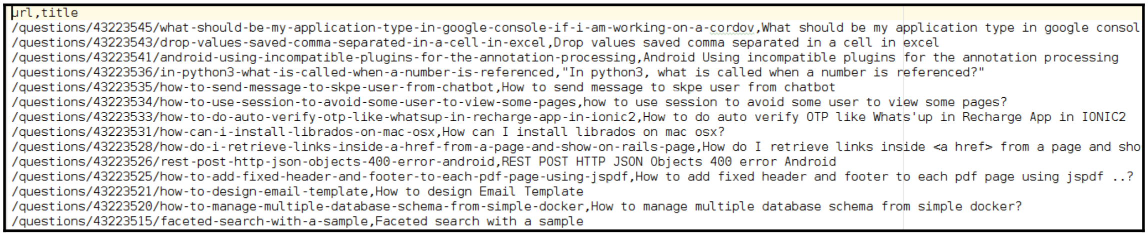
```
 % scrapy startproject web_scraping_test                                  NLPython/NLPython (master ⚡) jalaj-System-Product-Name
New Scrapy project 'web_scraping_test', using template directory '/usr/local/lib/python2.7/dist-packages/scrapy/templates/project', created in:
    /home/jalaj/PycharmProjects/NLPython/NLPython/web_scraping_test

You can start your first spider with:
    cd web_scraping_test
    scrapy genspider example example.com
 %                                                                        NLPython/NLPython (master ⚡) jalaj-System-Product-Name
```



```python
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class WebScrapingTestItem(scrapy.Item):
    title = scrapy.Field()
    url = scrapy.Field()
    pass
```

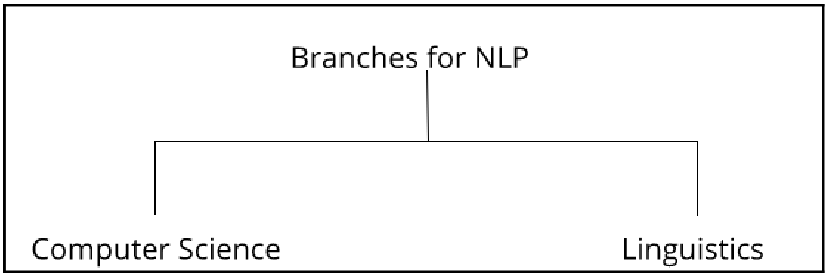


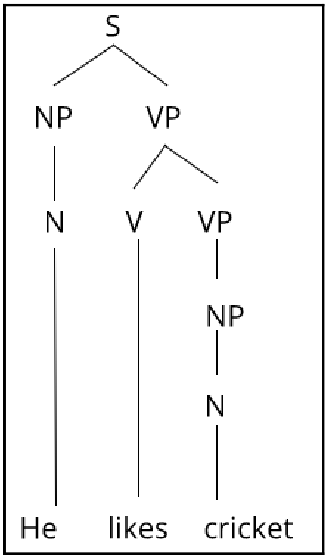
```python
from scrapy import Spider
from scrapy.selector import Selector

class WebScrapingTestspider(Spider):
    name = "WebScrapingTestspider"
    allowed_domains = ["stackoverflow.com"]
    start_urls = [
        "http://stackoverflow.com/questions?pagesize=50&sort=newest",
    ]

    def parse(self, response):
        questions = Selector(response).xpath('//div[@class="summary"]/h3')

        for question in questions:
            item = dict()
            item['title'] = question.xpath(
                'a[@class="question-hyperlink"]/text()').extract()[0]
            item['url'] = question.xpath(
                'a[@class="question-hyperlink"]/@href').extract()[0]
            yield item

#Now you can run this by using following commands.
#$ cd web_scraping_test/web_scraping_test
#If you wnat to export data in csv format execute the following command
#$ scrapy crawl WebScrapingTestspider -o result.csv -t csv
```

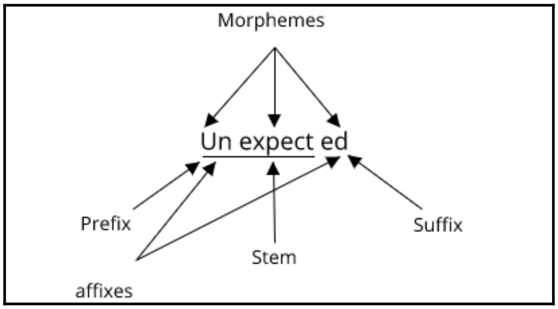


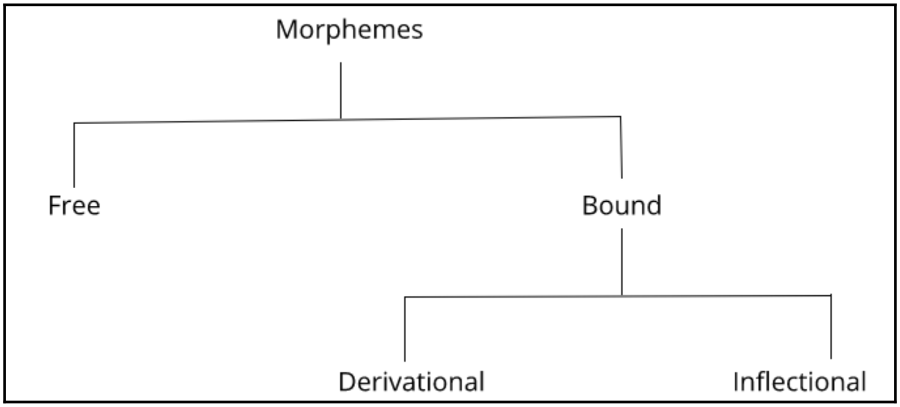
```
url,title
/questions/43223545/what-should-be-my-application-type-in-google-console-if-i-am-working-on-a-cordov,What should be my application type in google consol
/questions/43223543/drop-values-saved-comma-separated-in-a-cell-in-excel,Drop values saved comma separated in a cell in excel
/questions/43223541/android-using-incompatible-plugins-for-the-annotation-processing,Android Using incompatible plugins for the annotation processing
/questions/43223536/in-python3-what-is-called-when-a-number-is-referenced,"In python3, what is called when a number is referenced?"
/questions/43223535/how-to-send-message-to-skpe-user-from-chatbot,How to send message to skpe user from chatbot
/questions/43223534/how-to-use-session-to-avoid-some-user-to-view-some-pages,how to use session to avoid some user to view some pages?
/questions/43223533/how-to-do-auto-verify-otp-like-whatsup-in-recharge-app-in-ionic2,How to do auto verify OTP like Whats'up in Recharge App in IONIC2
/questions/43223531/how-can-i-install-librados-on-mac-osx,How can I install librados on mac osx?
/questions/43223528/how-do-i-retrieve-links-inside-a-href-from-a-page-and-show-on-rails-page,How do I retrieve links inside <a href> from a page and sho
/questions/43223526/rest-post-http-json-objects-400-error-android,REST POST HTTP JSON Objects 400 error Android
/questions/43223525/how-to-add-fixed-header-and-footer-to-each-pdf-page-using-jspdf,How to add fixed header and footer to each pdf page using jspdf ..?
/questions/43223521/how-to-design-email-template,How to design Email Template
/questions/43223520/how-to-manage-multiple-database-schema-from-simple-docker,How to manage multiple database schema from simple docker?
/questions/43223515/faceted-search-with-a-sample,Faceted search with a sample
```

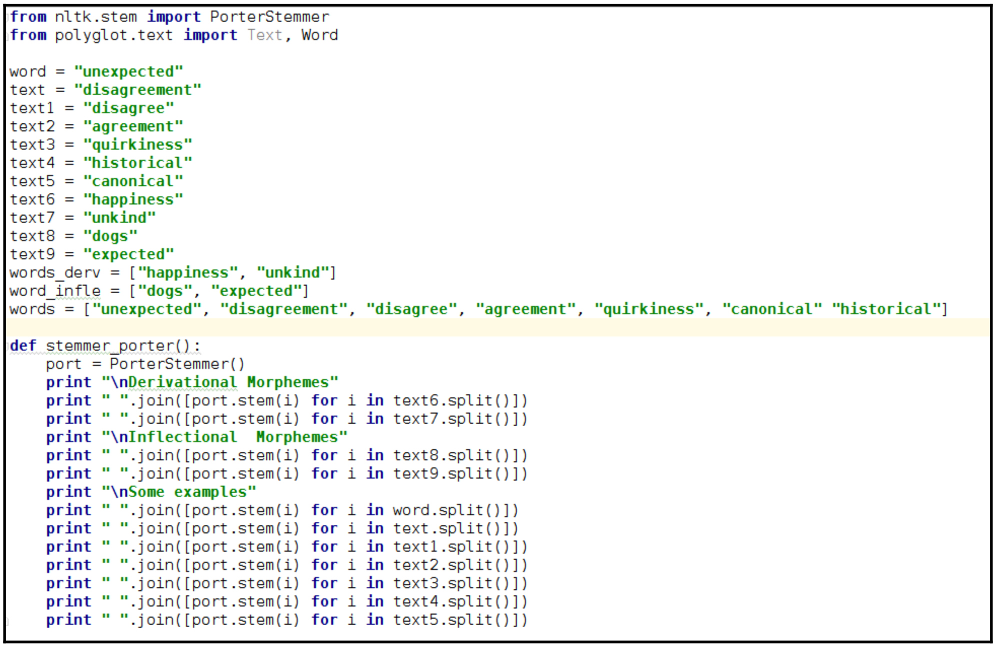
Chapter 3: Understanding Structure of Sentences



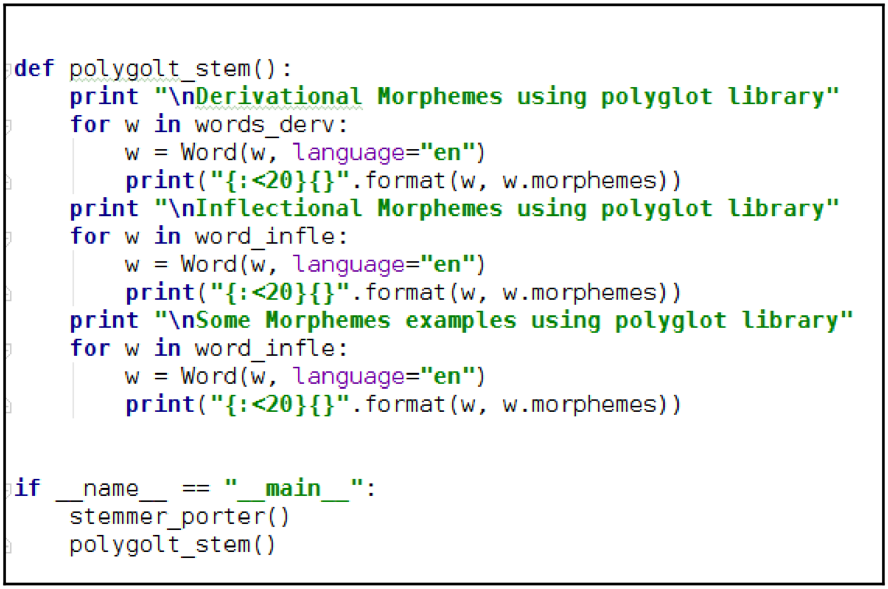




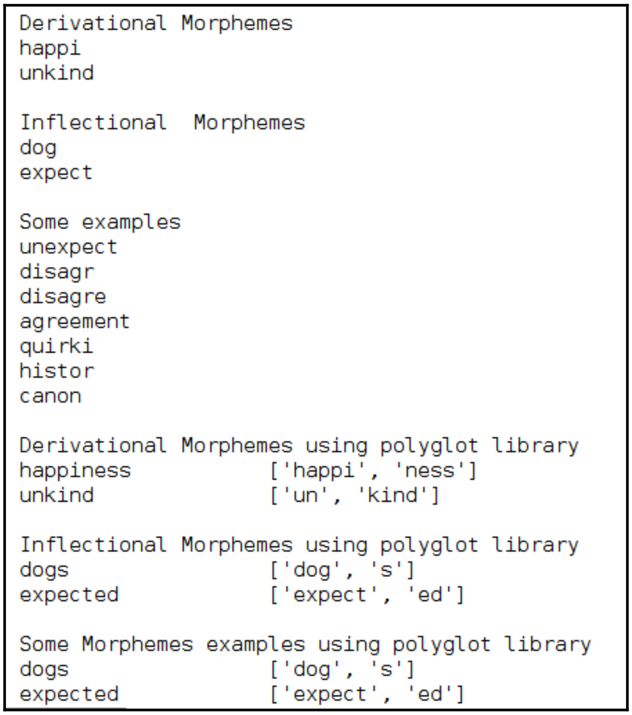




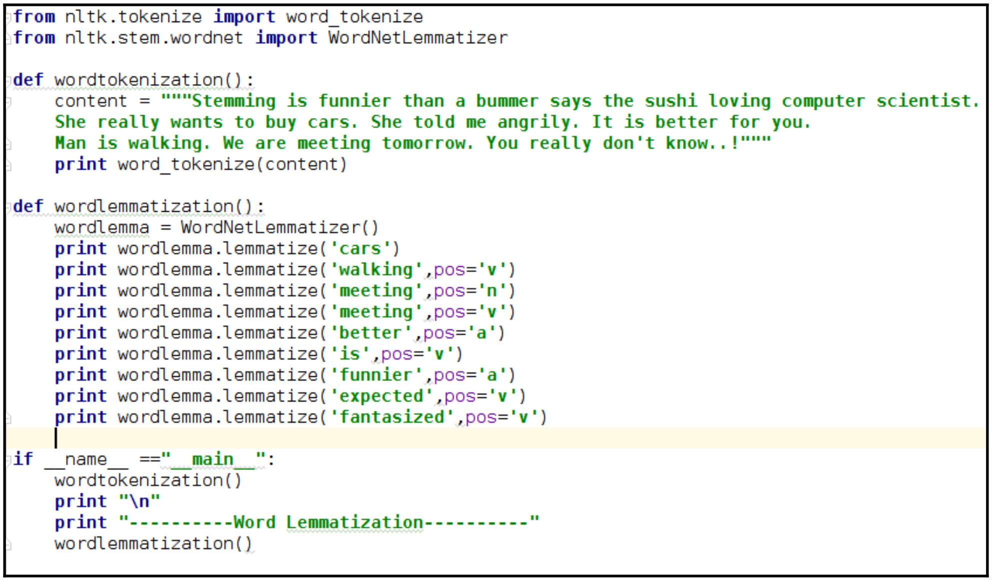
```
from nltk.stem import PorterStemmer
from polyglot.text import Text, Word

word = "unexpected"
text = "disagreement"
text1 = "disagree"
text2 = "agreement"
text3 = "quirkiness"
text4 = "historical"
text5 = "canonical"
text6 = "happiness"
text7 = "unkind"
text8 = "dogs"
text9 = "expected"
words_derv = ["happiness", "unkind"]
word_infle = ["dogs", "expected"]
words = ["unexpected", "disagreement", "disagree", "agreement", "quirkiness", "canonical" "historical"]

def stemmer_porter():
    port = PorterStemmer()
    print "\nDerivational Morphemes"
    print " ".join([port.stem(i) for i in text6.split()])
    print " ".join([port.stem(i) for i in text7.split()])
    print "\nInflectional  Morphemes"
    print " ".join([port.stem(i) for i in text8.split()])
    print " ".join([port.stem(i) for i in text9.split()])
    print "\nSome examples"
    print " ".join([port.stem(i) for i in word.split()])
    print " ".join([port.stem(i) for i in text.split()])
    print " ".join([port.stem(i) for i in text1.split()])
    print " ".join([port.stem(i) for i in text2.split()])
    print " ".join([port.stem(i) for i in text3.split()])
    print " ".join([port.stem(i) for i in text4.split()])
    print " ".join([port.stem(i) for i in text5.split()])
```



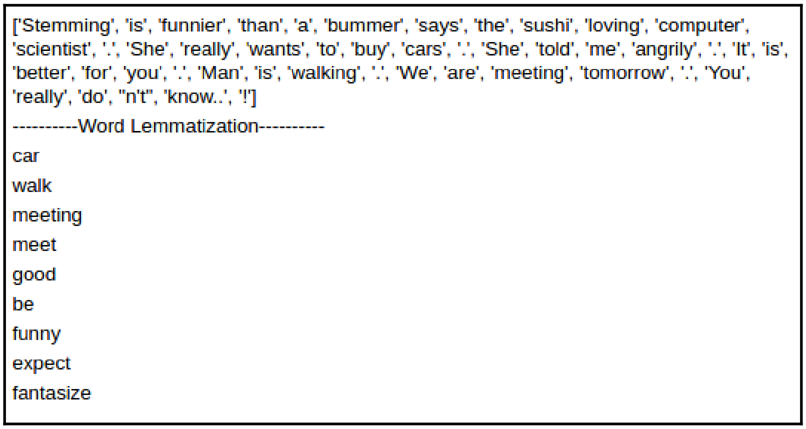
```
def polygolt_stem():
    print "\nDerivational Morphemes using polyglot library"
    for w in words_derv:
        w = Word(w, language="en")
        print("{:<20}{}".format(w, w.morphemes))
    print "\nInflectional Morphemes using polyglot library"
    for w in word_infle:
        w = Word(w, language="en")
        print("{:<20}{}".format(w, w.morphemes))
    print "\nSome Morphemes examples using polyglot library"
    for w in word_infle:
        w = Word(w, language="en")
        print("{:<20}{}".format(w, w.morphemes))


if __name__ == "__main__":
    stemmer_porter()
    polygolt_stem()
```



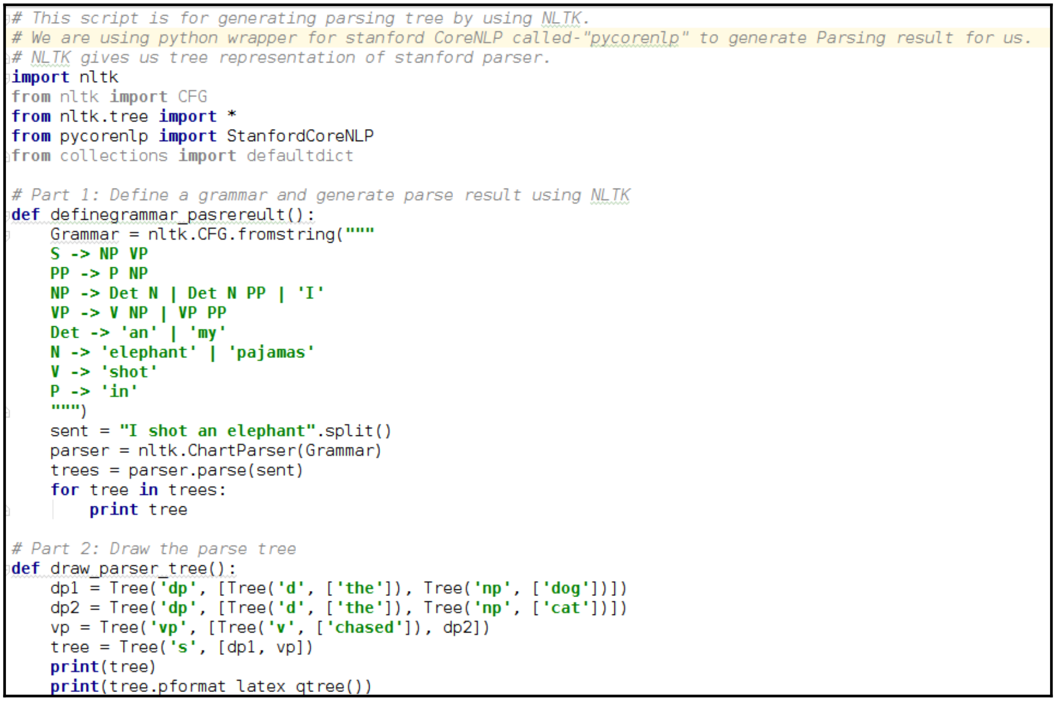
```
Derivational Morphemes
happi
unkind

Inflectional  Morphemes
dog
expect

Some examples
unexpect
disagr
disagre
agreement
quirki
histor
canon

Derivational Morphemes using polyglot library
happiness           ['happi', 'ness']
unkind              ['un', 'kind']

Inflectional Morphemes using polyglot library
dogs                ['dog', 's']
expected            ['expect', 'ed']

Some Morphemes examples using polyglot library
dogs                ['dog', 's']
expected            ['expect', 'ed']
```



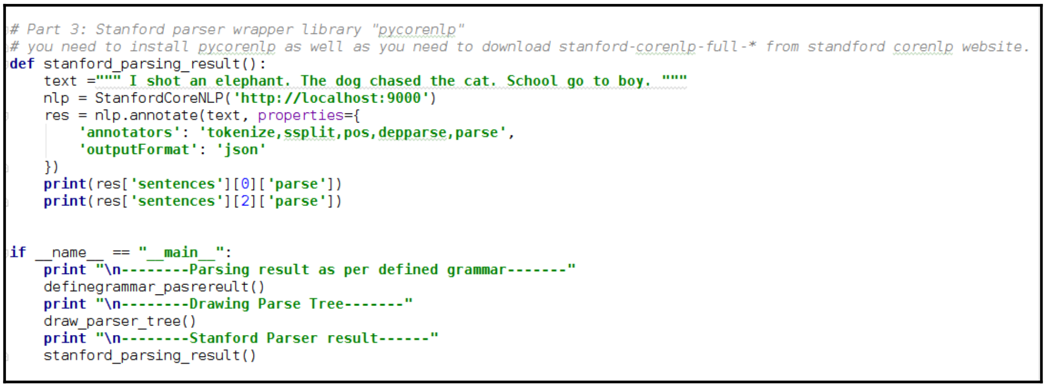
```
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer

def wordtokenization():
    content = """Stemming is funnier than a bummer says the sushi loving computer scientist.
    She really wants to buy cars. She told me angrily. It is better for you.
    Man is walking. We are meeting tomorrow. You really don't know..!"""
    print word_tokenize(content)

def wordlemmatization():
    wordlemma = WordNetLemmatizer()
    print wordlemma.lemmatize('cars')
    print wordlemma.lemmatize('walking',pos='v')
    print wordlemma.lemmatize('meeting',pos='n')
    print wordlemma.lemmatize('meeting',pos='v')
    print wordlemma.lemmatize('better',pos='a')
    print wordlemma.lemmatize('is',pos='v')
    print wordlemma.lemmatize('funnier',pos='a')
    print wordlemma.lemmatize('expected',pos='v')
    print wordlemma.lemmatize('fantasized',pos='v')

if __name__ =="__main__":
    wordtokenization()
    print "\n"
    print "----------Word Lemmatization----------"
    wordlemmatization()
```



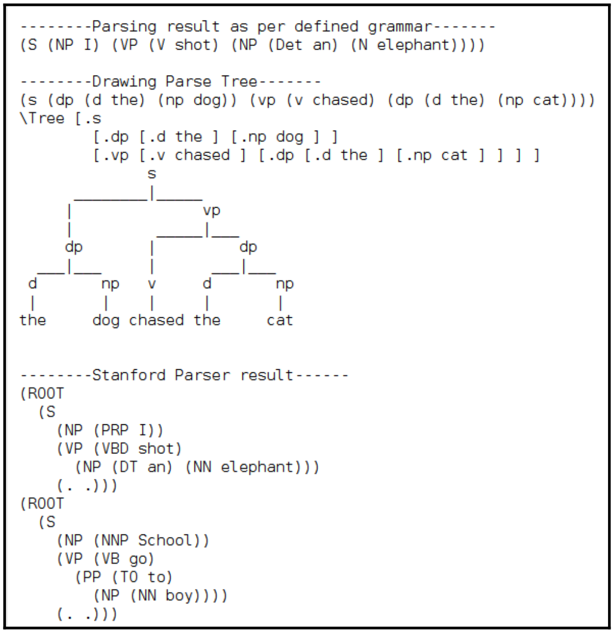
```
['Stemming', 'is', 'funnier', 'than', 'a', 'bummer', 'says', 'the', 'sushi', 'loving', 'computer',
'scientist', '.', 'She', 'really', 'wants', 'to', 'buy', 'cars', '.', 'She', 'told', 'me', 'angrily', '.', 'It', 'is',
'better', 'for', 'you', '.', 'Man', 'is', 'walking', '.', 'We', 'are', 'meeting', 'tomorrow', '.', 'You',
'really', 'do', "n't", 'know..', '!']
----------Word Lemmatization----------
car
walk
meeting
meet
good
be
funny
expect
fantasize
```



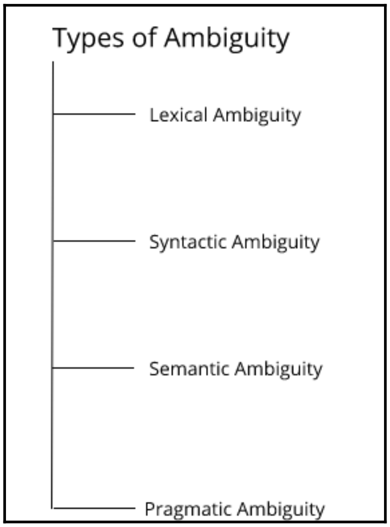
```
# This script is for generating parsing tree by using NLTK.
# We are using python wrapper for stanford CoreNLP called-"pycorenlp" to generate Parsing result for us.
# NLTK gives us tree representation of stanford parser.
import nltk
from nltk import CFG
from nltk.tree import *
from pycorenlp import StanfordCoreNLP
from collections import defaultdict

# Part 1: Define a grammar and generate parse result using NLTK
def definegrammar_pasrereult():
    Grammar = nltk.CFG.fromstring("""
    S -> NP VP
    PP -> P NP
    NP -> Det N | Det N PP | 'I'
    VP -> V NP | VP PP
    Det -> 'an' | 'my'
    N -> 'elephant' | 'pajamas'
    V -> 'shot'
    P -> 'in'
    """)
    sent = "I shot an elephant".split()
    parser = nltk.ChartParser(Grammar)
    trees = parser.parse(sent)
    for tree in trees:
        print tree

# Part 2: Draw the parse tree
def draw_parser_tree():
    dp1 = Tree('dp', [Tree('d', ['the']), Tree('np', ['dog'])])
    dp2 = Tree('dp', [Tree('d', ['the']), Tree('np', ['cat'])])
    vp = Tree('vp', [Tree('v', ['chased']), dp2])
    tree = Tree('s', [dp1, vp])
    print(tree)
    print(tree.pformat_latex_qtree())
```



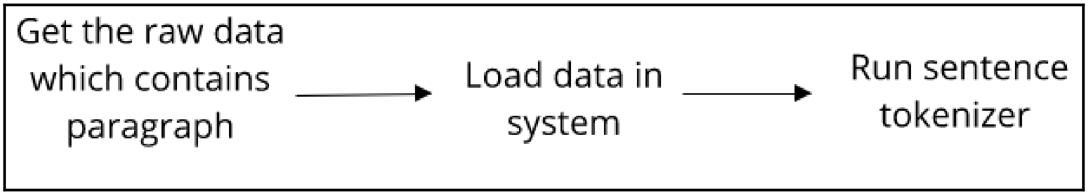
```
# Part 3: Stanford parser wrapper library "pycorenlp"
# you need to install pycorenlp as well as you need to download stanford-corenlp-full-* from standford corenlp website.
def stanford_parsing_result():
    text =""" I shot an elephant. The dog chased the cat. School go to boy. """
    nlp = StanfordCoreNLP('http://localhost:9000')
    res = nlp.annotate(text, properties={
        'annotators': 'tokenize,ssplit,pos,depparse,parse',
        'outputFormat': 'json'
    })
    print(res['sentences'][0]['parse'])
    print(res['sentences'][2]['parse'])


if __name__ == "__main__":
    print "\n--------Parsing result as per defined grammar-------"
    definegrammar_pasrereult()
    print "\n--------Drawing Parse Tree-------"
    draw_parser_tree()
    print "\n--------Stanford Parser result------"
    stanford_parsing_result()
```

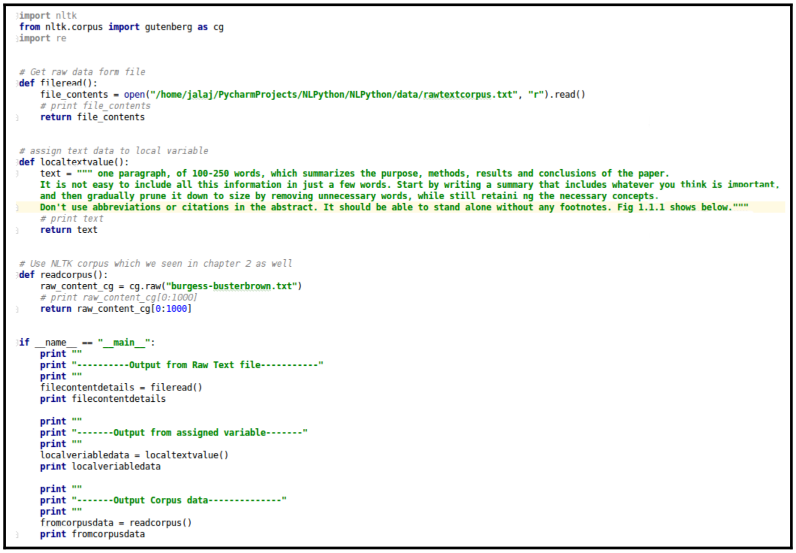


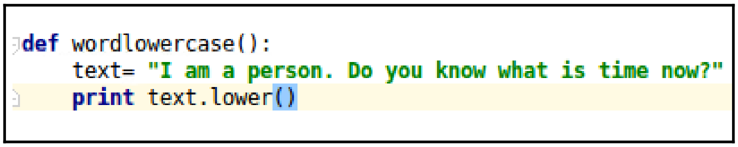
```
--------Parsing result as per defined grammar-------
(S (NP I) (VP (V shot) (NP (Det an) (N elephant))))

--------Drawing Parse Tree-------
(s (dp (d the) (np dog)) (vp (v chased) (dp (d the) (np cat))))
\Tree [.s
        [.dp [.d the ] [.np dog ] ]
        [.vp [.v chased ] [.dp [.d the ] [.np cat ] ] ] ]
              s
      ________|_____
     |              vp
     |         _____|___
     dp       |         dp
  ___|___     |      ___|___
 d       np   v     d       np
 |       |    |     |       |
the     dog chased the     cat


--------Stanford Parser result------
(ROOT
  (S
    (NP (PRP I))
    (VP (VBD shot)
      (NP (DT an) (NN elephant)))
    (. .)))
(ROOT
  (S
    (NP (NNP School))
    (VP (VB go)
      (PP (TO to)
        (NP (NN boy))))
    (. .)))
```

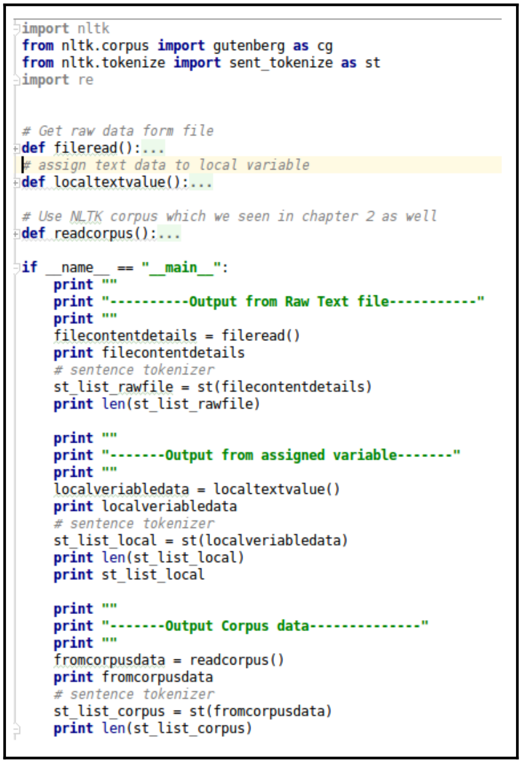


Chapter 4: Preprocessing

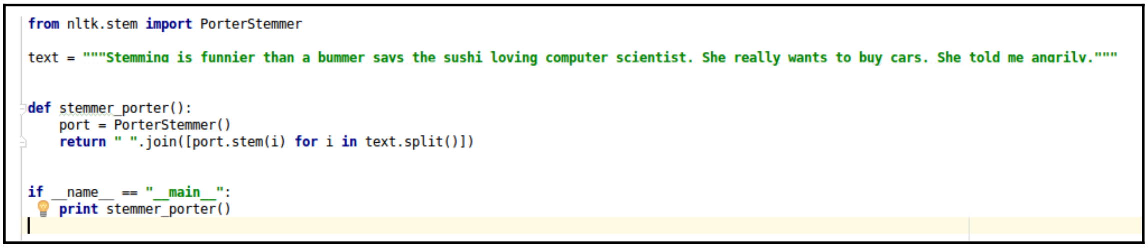

Get the raw data which contains paragraph → Load data in system → Run sentence tokenizer


```python
import nltk
from nltk.corpus import gutenberg as cg
import re


# Get raw data form file
def fileread():
    file_contents = open("/home/jalaj/PycharmProjects/NLPython/NLPython/data/rawtextcorpus.txt", "r").read()
    # print file_contents
    return file_contents


# assign text data to local variable
def localtextvalue():
    text = """ one paragraph, of 100-250 words, which summarizes the purpose, methods, results and conclusions of the paper.
    It is not easy to include all this information in just a few words. Start by writing a summary that includes whatever you think is important,
    and then gradually prune it down to size by removing unnecessary words, while still retaini ng the necessary concepts.
    Don't use abbreviations or citations in the abstract. It should be able to stand alone without any footnotes. Fig 1.1.1 shows below."""
    # print text
    return text


# Use NLTK corpus which we seen in chapter 2 as well
def readcorpus():
    raw_content_cg = cg.raw("burgess-busterbrown.txt")
    # print raw_content_cg[0:1000]
    return raw_content_cg[0:1000]


if __name__ == "__main__":
    print ""
    print "----------Output from Raw Text file-----------"
    print ""
    filecontentdetails = fileread()
    print filecontentdetails

    print ""
    print "-------Output from assigned variable-------"
    print ""
    localveriabledata = localtextvalue()
    print localveriabledata

    print ""
    print "-------Output Corpus data-------------"
    print ""
    fromcorpusdata = readcorpus()
    print fromcorpusdata
```

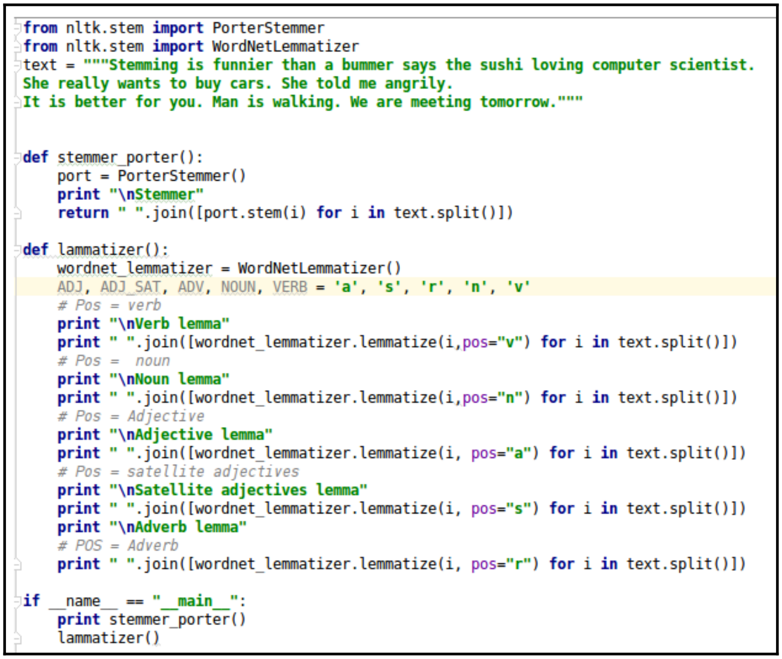

```python
def wordlowercase():
    text= "I am a person. Do you know what is time now?"
    print text.lower()
```



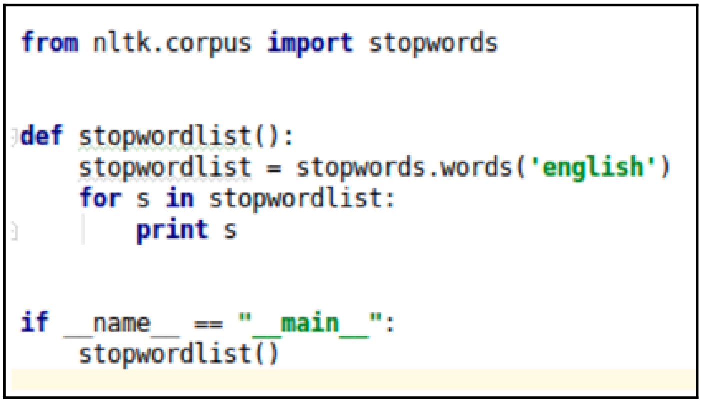
```
import nltk
from nltk.corpus import gutenberg as cg
from nltk.tokenize import sent_tokenize as st
import re


# Get raw data form file
def fileread():...
# assign text data to local variable
def localtextvalue():...

# Use NLTK corpus which we seen in chapter 2 as well
def readcorpus():...

if __name__ == "__main__":
    print ""
    print "----------Output from Raw Text file-----------"
    print ""
    filecontentdetails = fileread()
    print filecontentdetails
    # sentence tokenizer
    st_list_rawfile = st(filecontentdetails)
    print len(st_list_rawfile)

    print ""
    print "-------Output from assigned variable-------"
    print ""
    localveriabledata = localtextvalue()
    print localveriabledata
    # sentence tokenizer
    st_list_local = st(localveriabledata)
    print len(st_list_local)
    print st_list_local

    print ""
    print "-------Output Corpus data--------------"
    print ""
    fromcorpusdata = readcorpus()
    print fromcorpusdata
    # sentence tokenizer
    st_list_corpus = st(fromcorpusdata)
    print len(st_list_corpus)
```



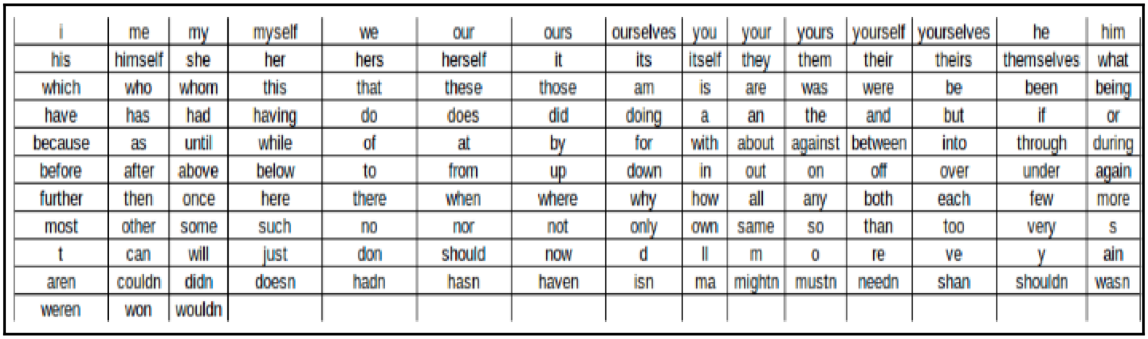
```
from nltk.stem import PorterStemmer

text = """Stemming is funnier than a bummer says the sushi loving computer scientist. She really wants to buy cars. She told me angrily."""


def stemmer_porter():
    port = PorterStemmer()
    return " ".join([port.stem(i) for i in text.split()])


if __name__ == "__main__":
    print stemmer_porter()
```



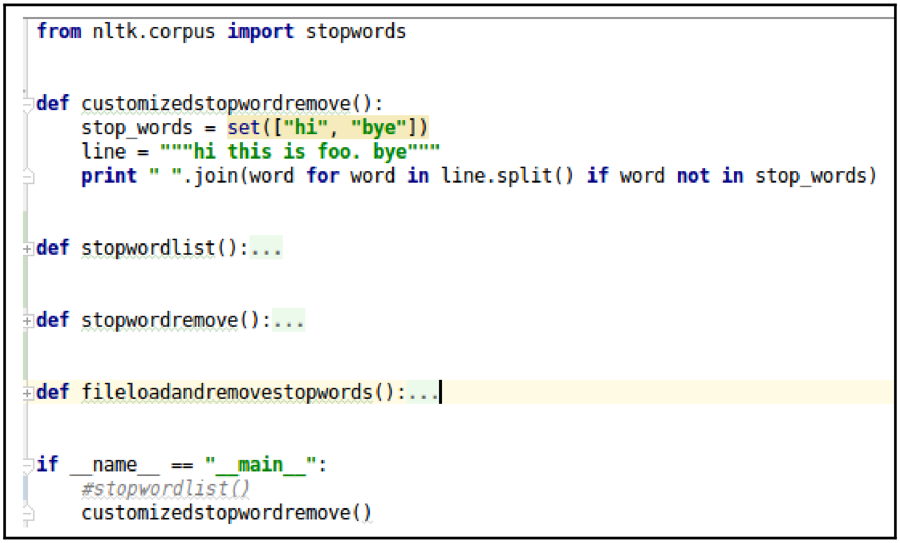
```
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
text = """Stemming is funnier than a bummer says the sushi loving computer scientist.
She really wants to buy cars. She told me angrily.
It is better for you. Man is walking. We are meeting tomorrow."""


def stemmer_porter():
    port = PorterStemmer()
    print "\nStemmer"
    return " ".join([port.stem(i) for i in text.split()])

def lammatizer():
    wordnet_lemmatizer = WordNetLemmatizer()
    ADJ, ADJ_SAT, ADV, NOUN, VERB = 'a', 's', 'r', 'n', 'v'
    # Pos = verb
    print "\nVerb lemma"
    print " ".join([wordnet_lemmatizer.lemmatize(i,pos="v") for i in text.split()])
    # Pos =  noun
    print "\nNoun lemma"
    print " ".join([wordnet_lemmatizer.lemmatize(i,pos="n") for i in text.split()])
    # Pos = Adjective
    print "\nAdjective lemma"
    print " ".join([wordnet_lemmatizer.lemmatize(i, pos="a") for i in text.split()])
    # Pos = satellite adjectives
    print "\nSatellite adjectives lemma"
    print " ".join([wordnet_lemmatizer.lemmatize(i, pos="s") for i in text.split()])
    print "\nAdverb lemma"
    # POS = Adverb
    print " ".join([wordnet_lemmatizer.lemmatize(i, pos="r") for i in text.split()])

if __name__ == "__main__":
    print stemmer_porter()
    lammatizer()
```



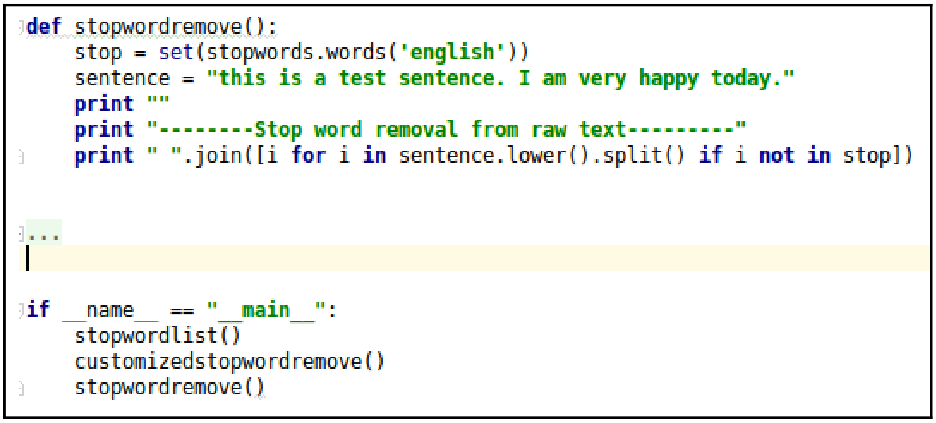
```
from nltk.corpus import stopwords


def stopwordlist():
    stopwordlist = stopwords.words('english')
    for s in stopwordlist:
        print s


if __name__ == "__main__":
    stopwordlist()
```



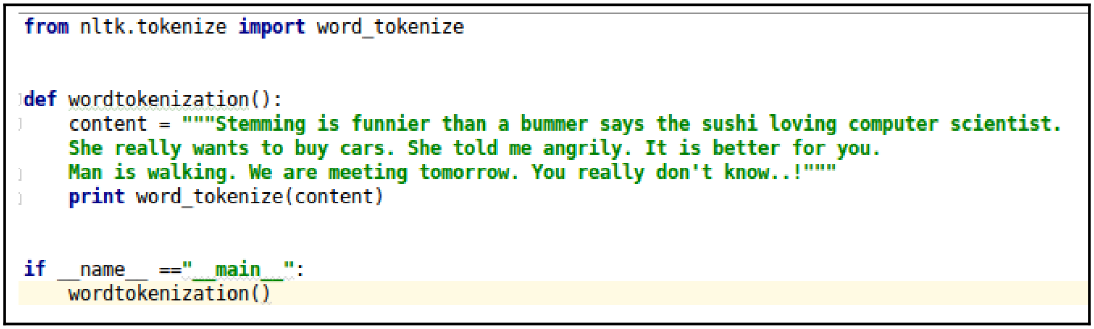
i	me	my	myself	we	our	ours	ourselves	you	your	yours	yourself	yourselves	he	him
his	himself	she	her	hers	herself	it	its	itself	they	them	their	theirs	themselves	what
which	who	whom	this	that	these	those	am	is	are	was	were	be	been	being
have	has	had	having	do	does	did	doing	a	an	the	and	but	if	or
because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



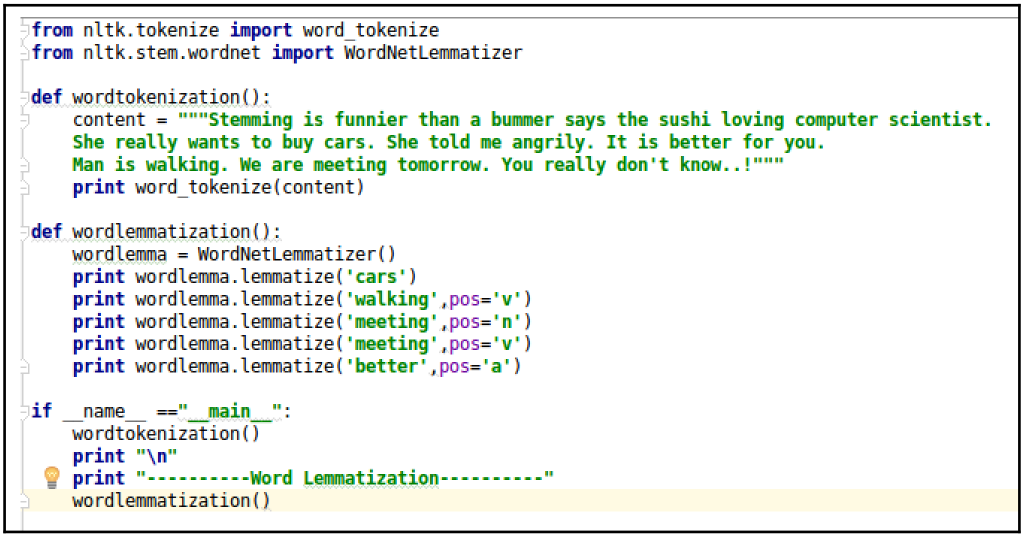
```
from nltk.corpus import stopwords


def customizedstopwordremove():
    stop_words = set(["hi", "bye"])
    line = """hi this is foo. bye"""
    print " ".join(word for word in line.split() if word not in stop_words)


def stopwordlist():...


def stopwordremove():...


def fileloadandremovestopwords():...


if __name__ == "__main__":
    #stopwordlist()
    customizedstopwordremove()
```



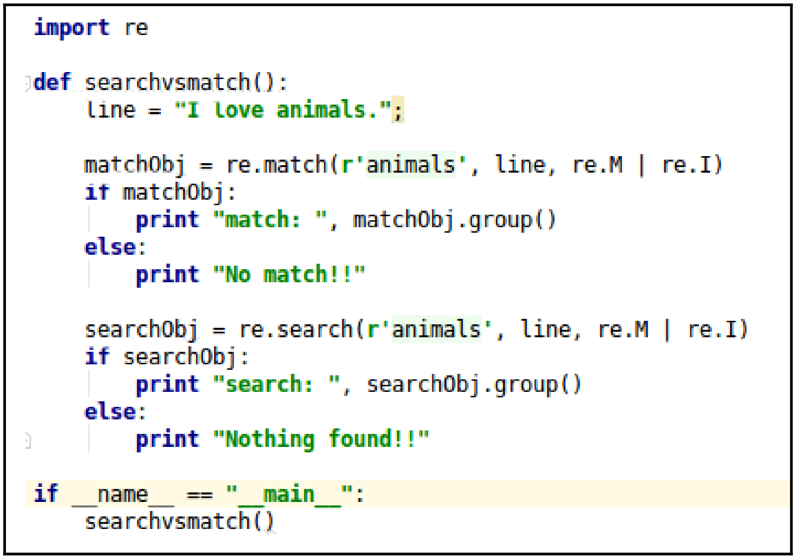
```
def stopwordremove():
    stop = set(stopwords.words('english'))
    sentence = "this is a test sentence. I am very happy today."
    print ""
    print "--------Stop word removal from raw text---------"
    print " ".join([i for i in sentence.lower().split() if i not in stop])


...


if __name__ == "__main__":
    stopwordlist()
    customizedstopwordremove()
    stopwordremove()
```



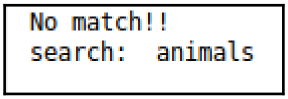
```
from nltk.tokenize import word_tokenize


def wordtokenization():
    content = """Stemming is funnier than a bummer says the sushi loving computer scientist.
    She really wants to buy cars. She told me angrily. It is better for you.
    Man is walking. We are meeting tomorrow. You really don't know..!"""
    print word_tokenize(content)


if __name__ =="__main__":
    wordtokenization()
```



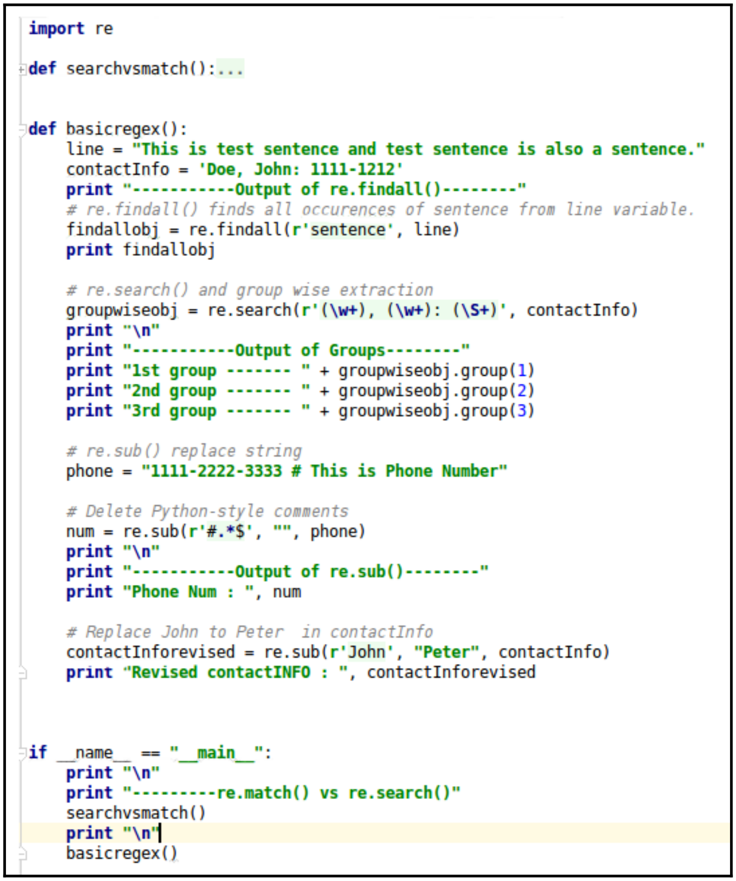
```
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer

def wordtokenization():
    content = """Stemming is funnier than a bummer says the sushi loving computer scientist.
    She really wants to buy cars. She told me angrily. It is better for you.
    Man is walking. We are meeting tomorrow. You really don't know..!"""
    print word_tokenize(content)

def wordlemmatization():
    wordlemma = WordNetLemmatizer()
    print wordlemma.lemmatize('cars')
    print wordlemma.lemmatize('walking',pos='v')
    print wordlemma.lemmatize('meeting',pos='n')
    print wordlemma.lemmatize('meeting',pos='v')
    print wordlemma.lemmatize('better',pos='a')

if __name__ =="__main__":
    wordtokenization()
    print "\n"
    print "----------Word Lemmatization----------"
    wordlemmatization()
```



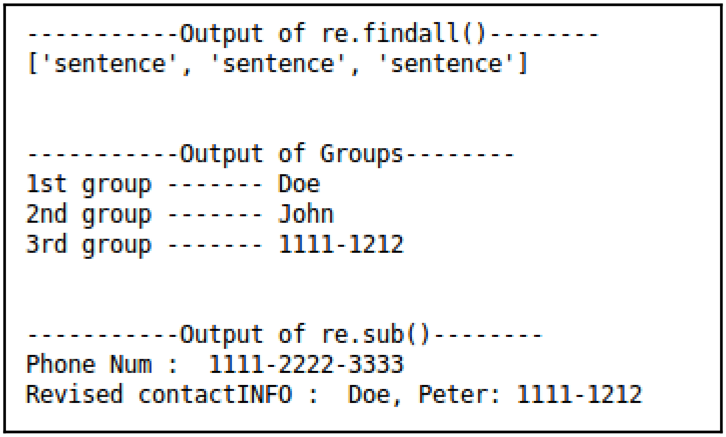
```
import re

def searchvsmatch():
    line = "I love animals.";

    matchObj = re.match(r'animals', line, re.M | re.I)
    if matchObj:
        print "match: ", matchObj.group()
    else:
        print "No match!!"

    searchObj = re.search(r'animals', line, re.M | re.I)
    if searchObj:
        print "search: ", searchObj.group()
    else:
        print "Nothing found!!"

if __name__ == "__main__":
    searchvsmatch()
```



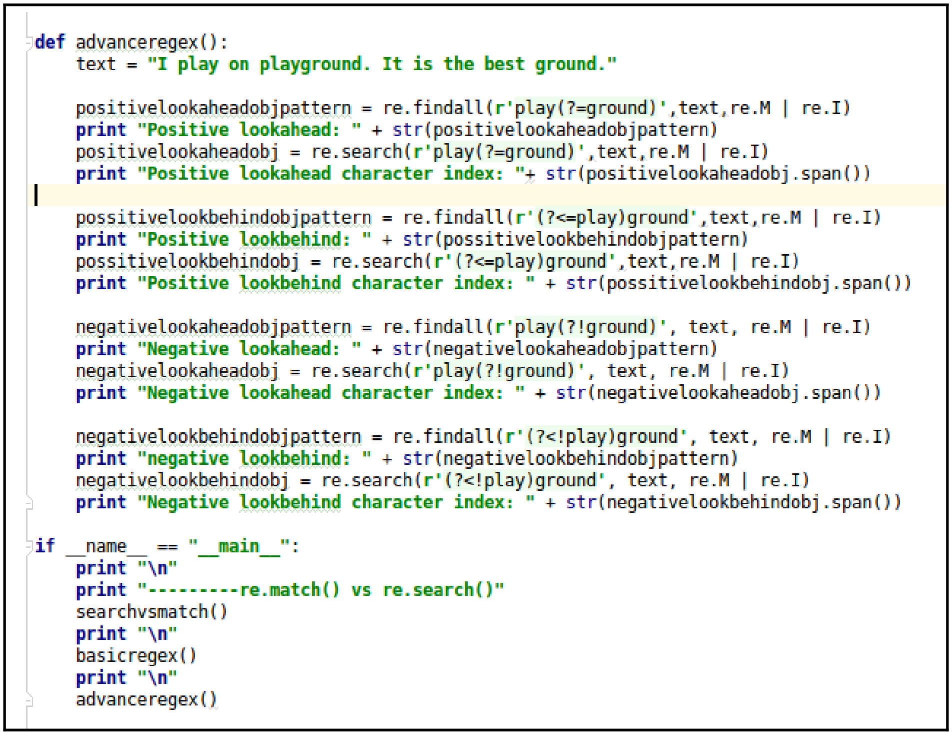
```
No match!!
search:  animals
```



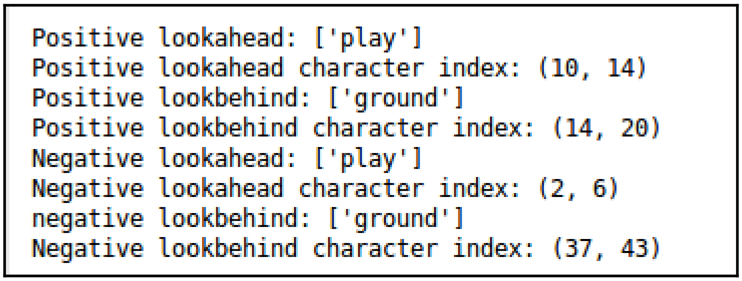
```python
import re

def searchvsmatch():...


def basicregex():
    line = "This is test sentence and test sentence is also a sentence."
    contactInfo = 'Doe, John: 1111-1212'
    print "-----------Output of re.findall()--------"
    # re.findall() finds all occurences of sentence from line variable.
    findallobj = re.findall(r'sentence', line)
    print findallobj

    # re.search() and group wise extraction
    groupwiseobj = re.search(r'(\w+), (\w+): (\S+)', contactInfo)
    print "\n"
    print "-----------Output of Groups--------"
    print "1st group ------- " + groupwiseobj.group(1)
    print "2nd group ------- " + groupwiseobj.group(2)
    print "3rd group ------- " + groupwiseobj.group(3)

    # re.sub() replace string
    phone = "1111-2222-3333 # This is Phone Number"

    # Delete Python-style comments
    num = re.sub(r'#.*$', "", phone)
    print "\n"
    print "-----------Output of re.sub()--------"
    print "Phone Num : ", num

    # Replace John to Peter  in contactInfo
    contactInforevised = re.sub(r'John', "Peter", contactInfo)
    print "Revised contactINFO : ", contactInforevised



if __name__ == "__main__":
    print "\n"
    print "---------re.match() vs re.search()"
    searchvsmatch()
    print "\n"
    basicregex()
```



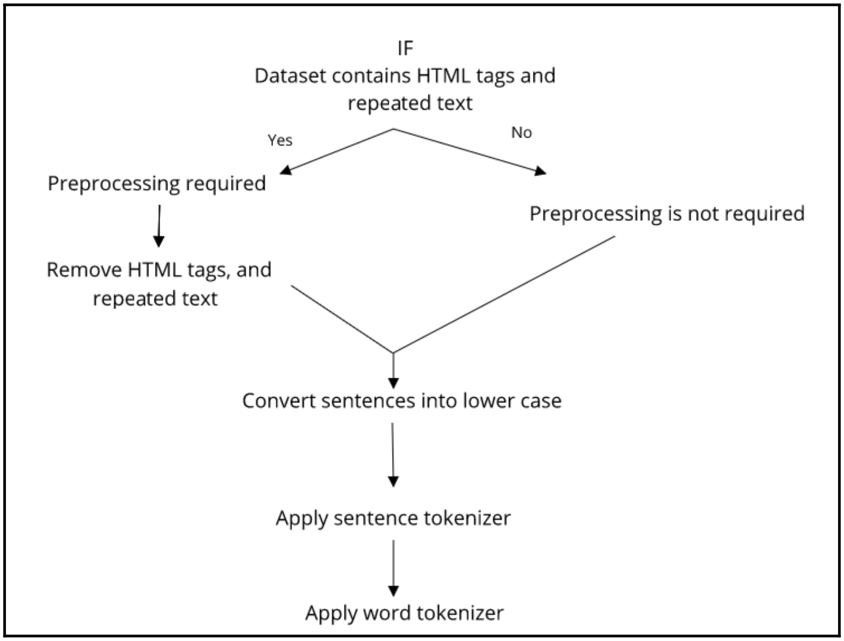
```
-----------Output of re.findall()--------
['sentence', 'sentence', 'sentence']


-----------Output of Groups--------
1st group ------- Doe
2nd group ------- John
3rd group ------- 1111-1212


-----------Output of re.sub()--------
Phone Num :  1111-2222-3333
Revised contactINFO :  Doe, Peter: 1111-1212
```



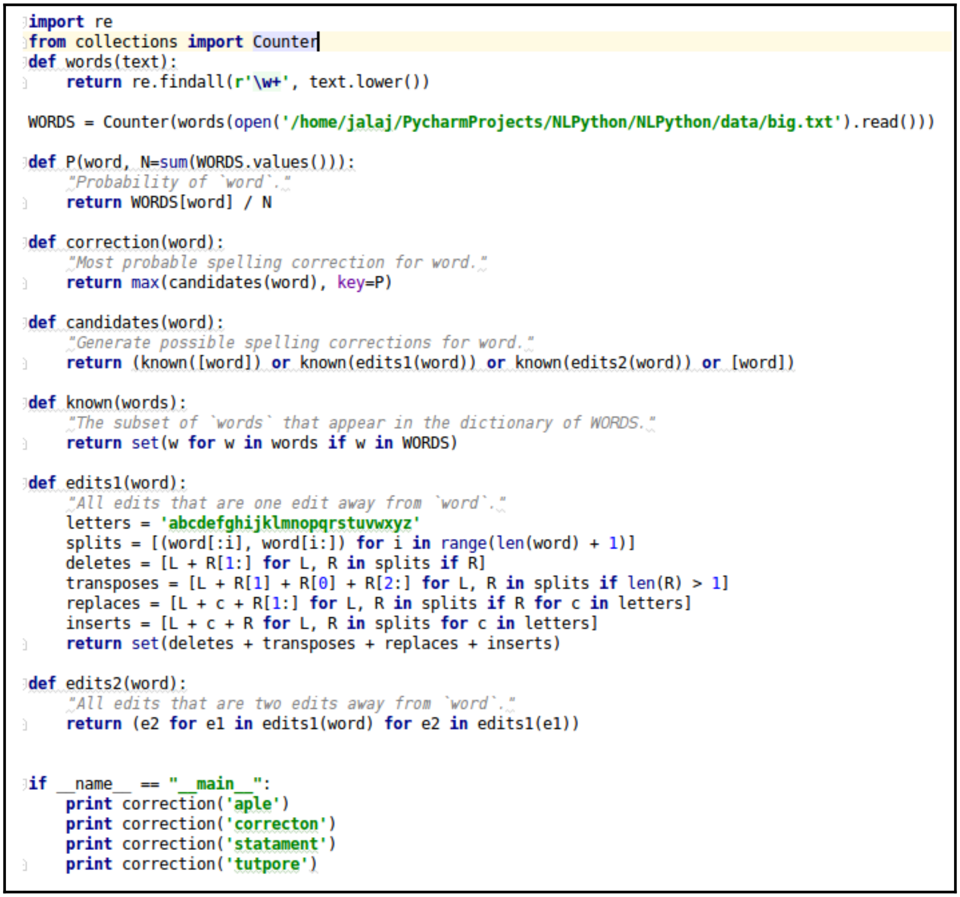
```
def advanceregex():
    text = "I play on playground. It is the best ground."

    positivelookaheadobjpattern = re.findall(r'play(?=ground)',text,re.M | re.I)
    print "Positive lookahead: " + str(positivelookaheadobjpattern)
    positivelookaheadobj = re.search(r'play(?=ground)',text,re.M | re.I)
    print "Positive lookahead character index: "+ str(positivelookaheadobj.span())

    possitivelookbehindobjpattern = re.findall(r'(?<=play)ground',text,re.M | re.I)
    print "Positive lookbehind: " + str(possitivelookbehindobjpattern)
    possitivelookbehindobj = re.search(r'(?<=play)ground',text,re.M | re.I)
    print "Positive lookbehind character index: " + str(possitivelookbehindobj.span())

    negativelookaheadobjpattern = re.findall(r'play(?!ground)', text, re.M | re.I)
    print "Negative lookahead: " + str(negativelookaheadobjpattern)
    negativelookaheadobj = re.search(r'play(?!ground)', text, re.M | re.I)
    print "Negative lookahead character index: " + str(negativelookaheadobj.span())

    negativelookbehindobjpattern = re.findall(r'(?<!play)ground', text, re.M | re.I)
    print "negative lookbehind: " + str(negativelookbehindobjpattern)
    negativelookbehindobj = re.search(r'(?<!play)ground', text, re.M | re.I)
    print "Negative lookbehind character index: " + str(negativelookbehindobj.span())

if __name__ == "__main__":
    print "\n"
    print "---------re.match() vs re.search()"
    searchvsmatch()
    print "\n"
    basicregex()
    print "\n"
    advanceregex()
```



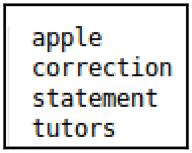
```
Positive lookahead: ['play']
Positive lookahead character index: (10, 14)
Positive lookbehind: ['ground']
Positive lookbehind character index: (14, 20)
Negative lookahead: ['play']
Negative lookahead character index: (2, 6)
negative lookbehind: ['ground']
Negative lookbehind character index: (37, 43)
```


IF
Dataset contains HTML tags and repeated text
Yes
No
Preprocessing required
Preprocessing is not required
Remove HTML tags, and repeated text
Convert sentences into lower case
Apply sentence tokenizer
Apply word tokenizer

	#	t	u	t	o	u	r
#	0	1	2	3	4	5	6
t	1	0	1	2	3	4	5
u	2	1	0	1	2	3	4
t	3	2	1	1	2	3	4
o	4	3	2	2	1	2	3
r	5	4	3	3	2	2	2

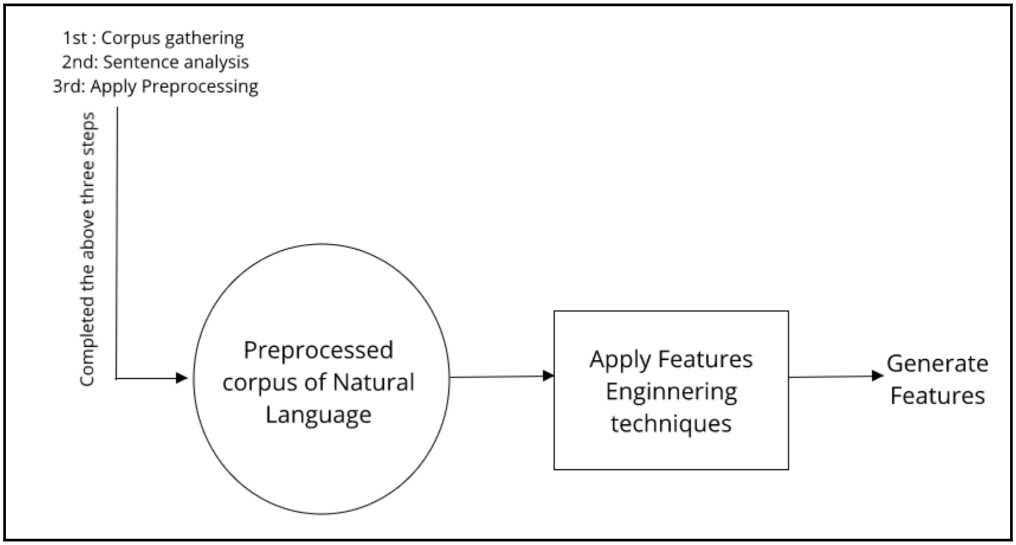


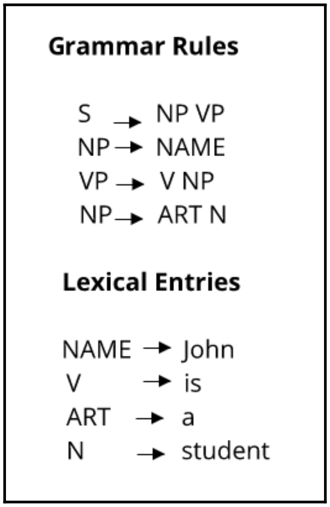
```
import re
from collections import Counter
def words(text):
    return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('/home/jalaj/PycharmProjects/NLPython/NLPython/data/big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    letters = 'abcdefghijklmnopqrstuvwxyz'
    splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes = [L + R[1:] for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R) > 1]
    replaces = [L + c + R[1:] for L, R in splits if R for c in letters]
    inserts = [L + c + R for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))


if __name__ == "__main__":
    print correction('aple')
    print correction('correcton')
    print correction('statament')
    print correction('tutpore')
```



```
apple
correction
statement
tutors
```

Chapter 5: Feature Engineering and NLP Algorithms


1st : Corpus gathering
2nd: Sentence analysis
3rd: Apply Preprocessing
Completed the above three steps
Preprocessed corpus of Natural Language
Apply Features Enginnering techniques
Generate Features

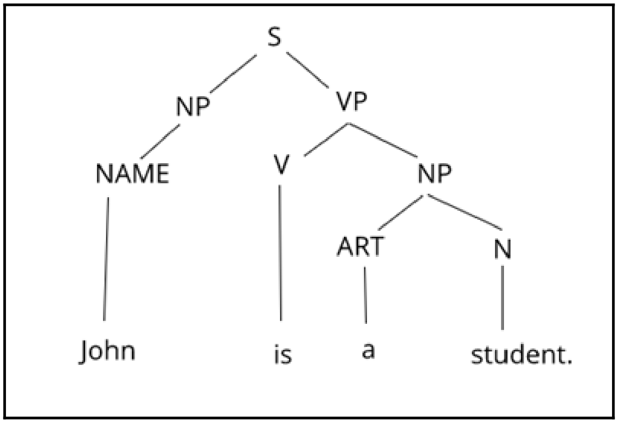


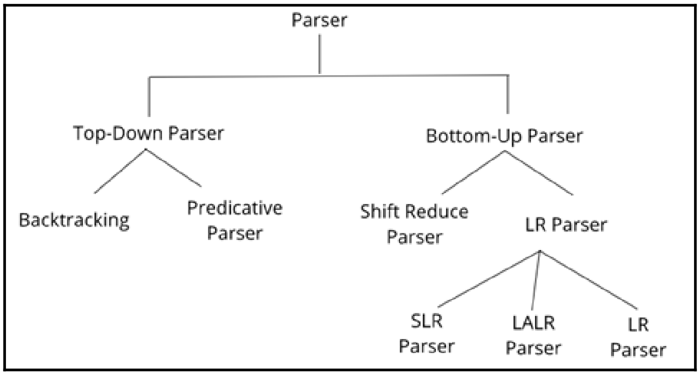
Grammar Rules

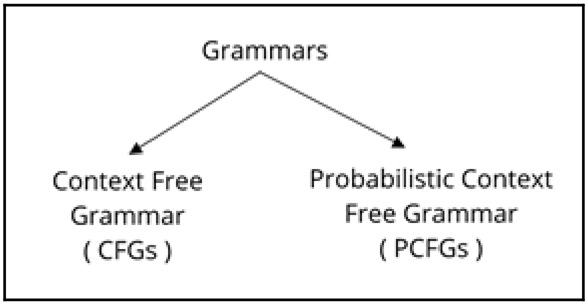
S → NP VP
NP → NAME
VP → V NP
NP → ART N

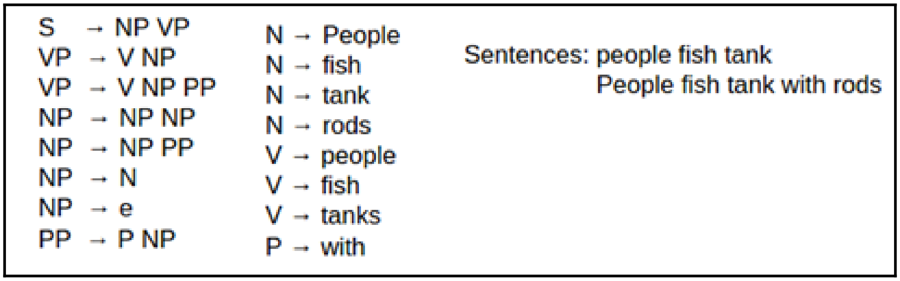
Lexical Entries

NAME → John
V → is
ART → a
N → student


S
NP
VP
NAME
V
NP
ART
N
John
is
a
student.


Parser
Top-Down Parser
Bottom-Up Parser
Backtracking
Predicative Parser
Shift Reduce Parser
LR Parser
SLR Parser
LALR Parser
LR Parser

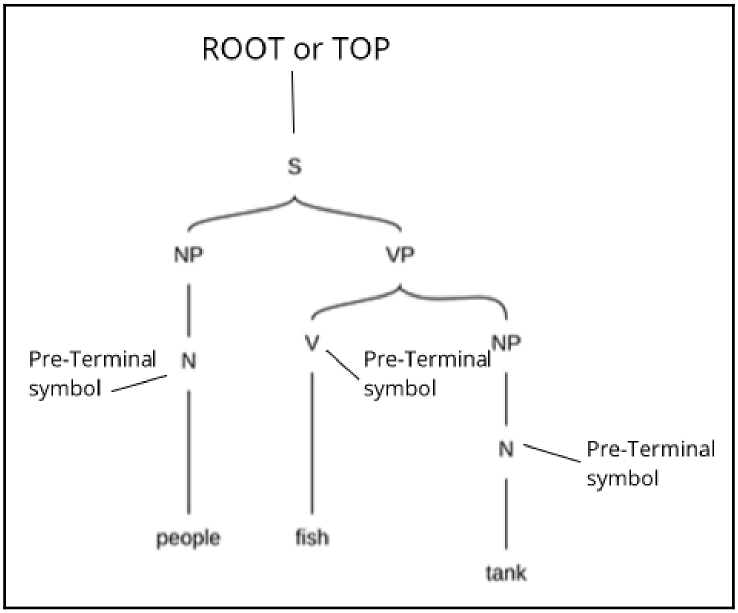

Grammars
Context Free Grammar (CFGs)
Probabilistic Context Free Grammar (PCFGs)

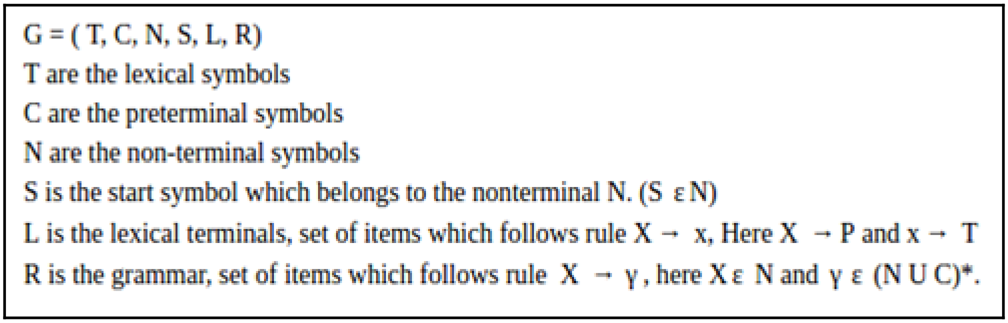


S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

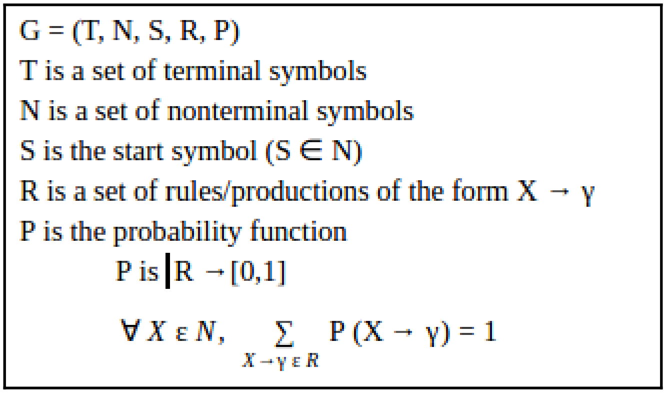
N → People
N → fish
N → tank
N → rods
V → people
V → fish
V → tanks
P → with

Sentences: people fish tank
People fish tank with rods





G = (T, C, N, S, L, R)
T are the lexical symbols
C are the preterminal symbols
N are the non-terminal symbols
S is the start symbol which belongs to the nonterminal N. (S ε N)
L is the lexical terminals, set of items which follows rule X → x, Here X → P and x → T
R is the grammar, set of items which follows rule X → γ , here X ε N and γ ε (N U C)*.



G = (T, N, S, R, P)

T is a set of terminal symbols

N is a set of nonterminal symbols

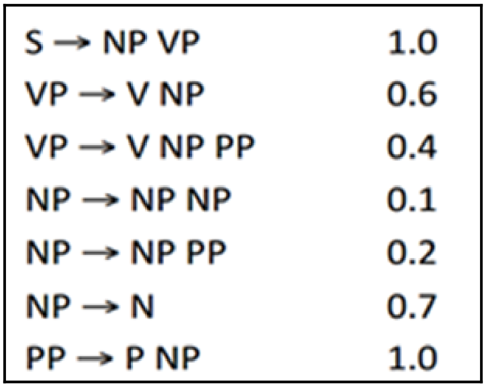
S is the start symbol (S ∈ N)

R is a set of rules/productions of the form $X \to \gamma$

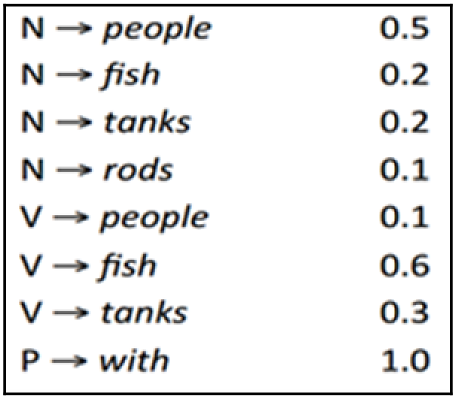
P is the probability function

P is R → [0,1]

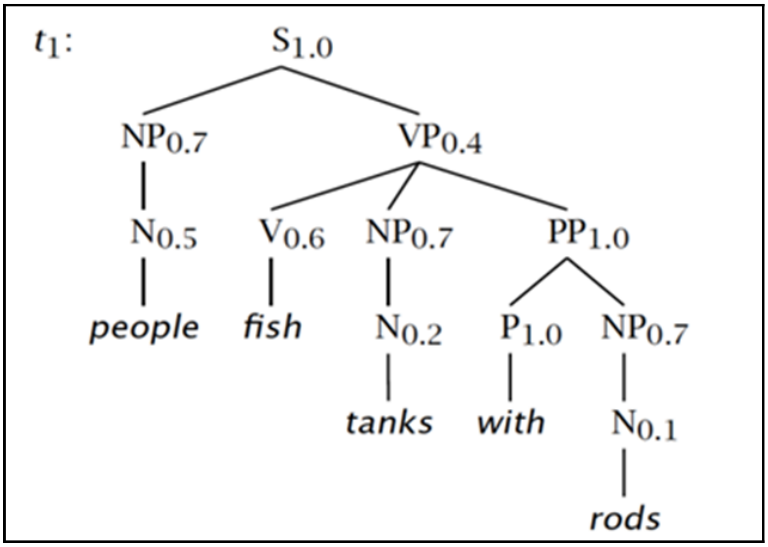
$$\forall X \,\varepsilon\, N, \quad \sum_{X \to \gamma \,\varepsilon\, R} P(X \to \gamma) = 1$$

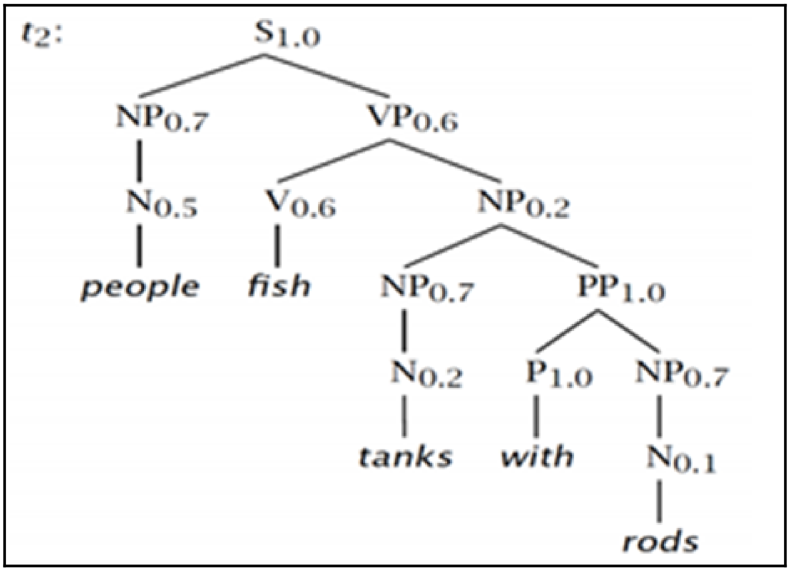


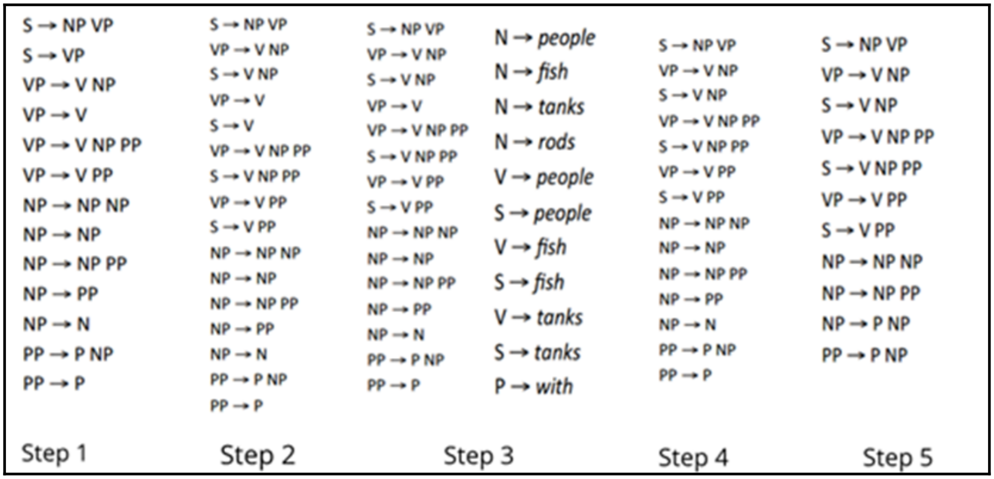
S → NP VP	1.0
VP → V NP	0.6
VP → V NP PP	0.4
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0



N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0


t_1:
$S_{1.0}$
$NP_{0.7}$
$VP_{0.4}$
$N_{0.5}$
$V_{0.6}$
$NP_{0.7}$
$PP_{1.0}$
people
fish
$N_{0.2}$
$P_{1.0}$
$NP_{0.7}$
tanks
with
$N_{0.1}$
rods


t_2:
$S_{1.0}$
$NP_{0.7}$
$VP_{0.6}$
$N_{0.5}$
$V_{0.6}$
$NP_{0.2}$
people
fish
$NP_{0.7}$
$PP_{1.0}$
$N_{0.2}$
$P_{1.0}$
$NP_{0.7}$
tanks
with
$N_{0.1}$
rods



S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Step 1

S → NP VP
VP → V NP
S → V NP
VP → V
S → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Step 2

S → NP VP
VP → V NP
S → V NP
VP → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
V → *fish*
S → *fish*
V → *tanks*
S → *tanks*
P → *with*

Step 3

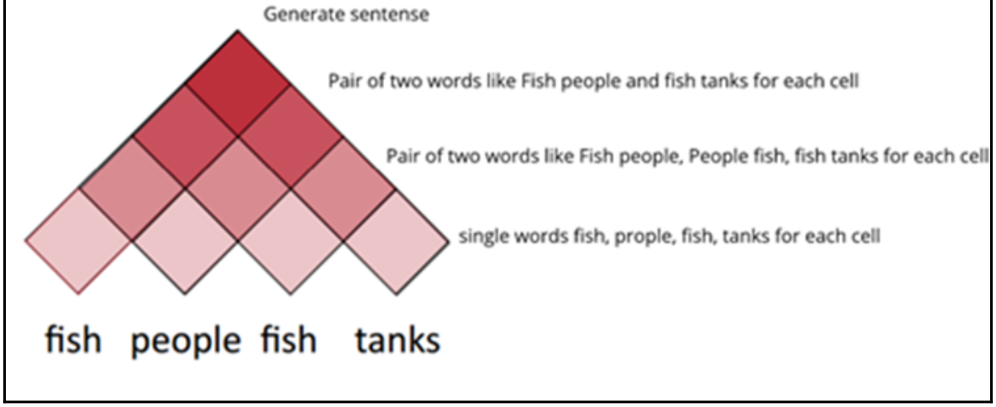
S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

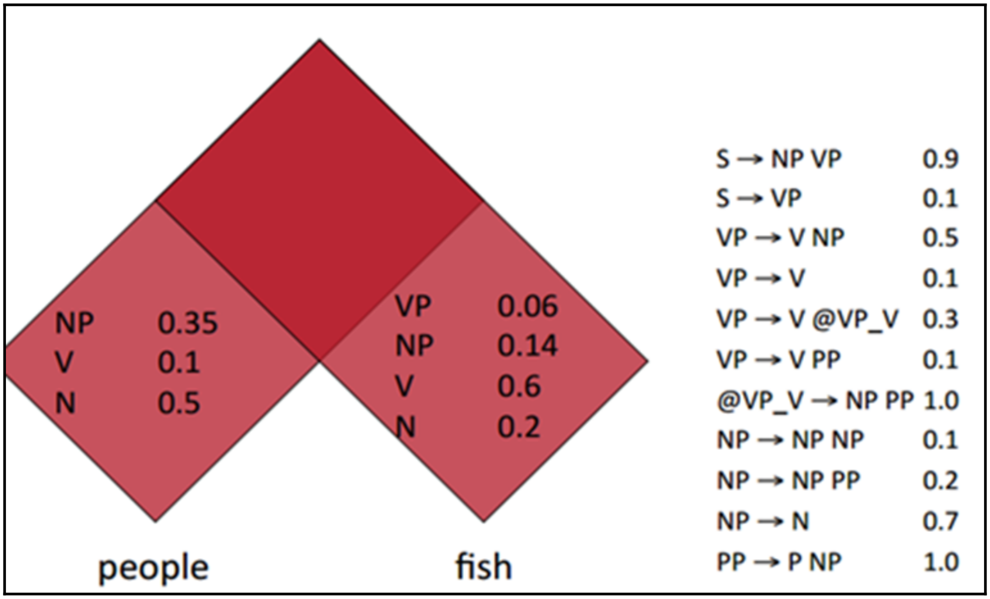
Step 4

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

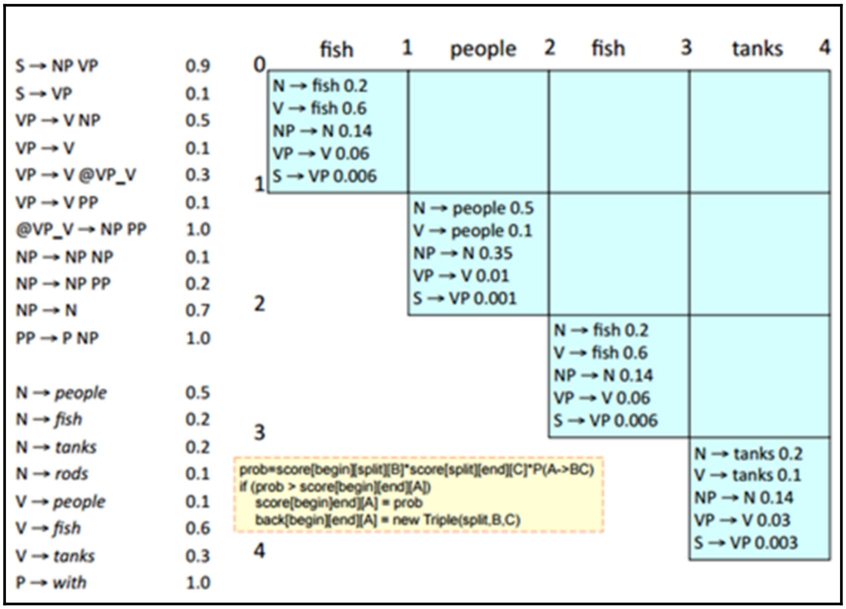
Step 5

Parse Triangle or Chart


Generate sentense
Pair of two words like Fish people and fish tanks for each cell
Pair of two words like Fish people, People fish, fish tanks for each cell
single words fish, prople, fish, tanks for each cell
fish
people
fish
tanks



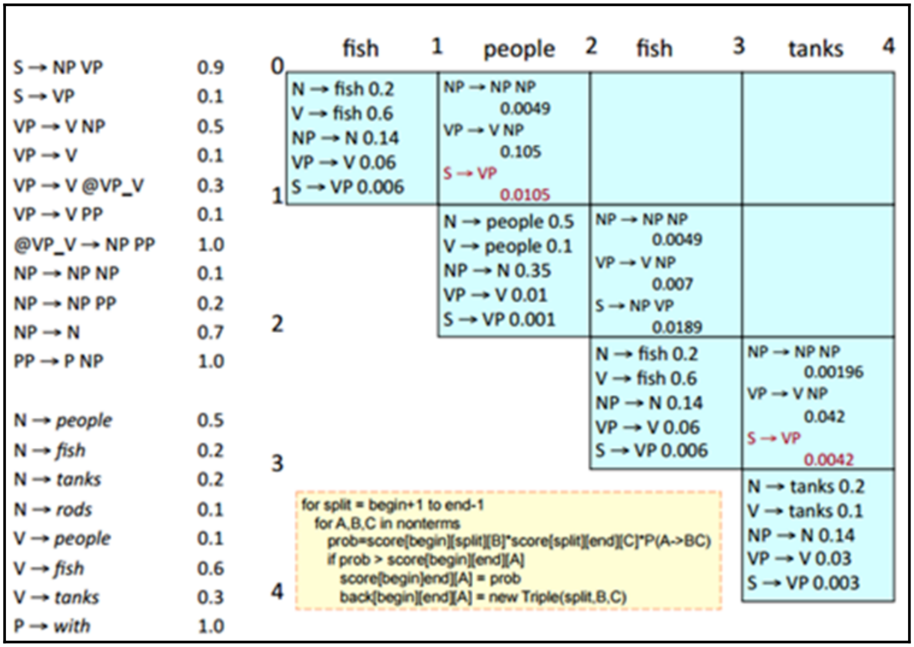
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0



S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006			
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001		
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

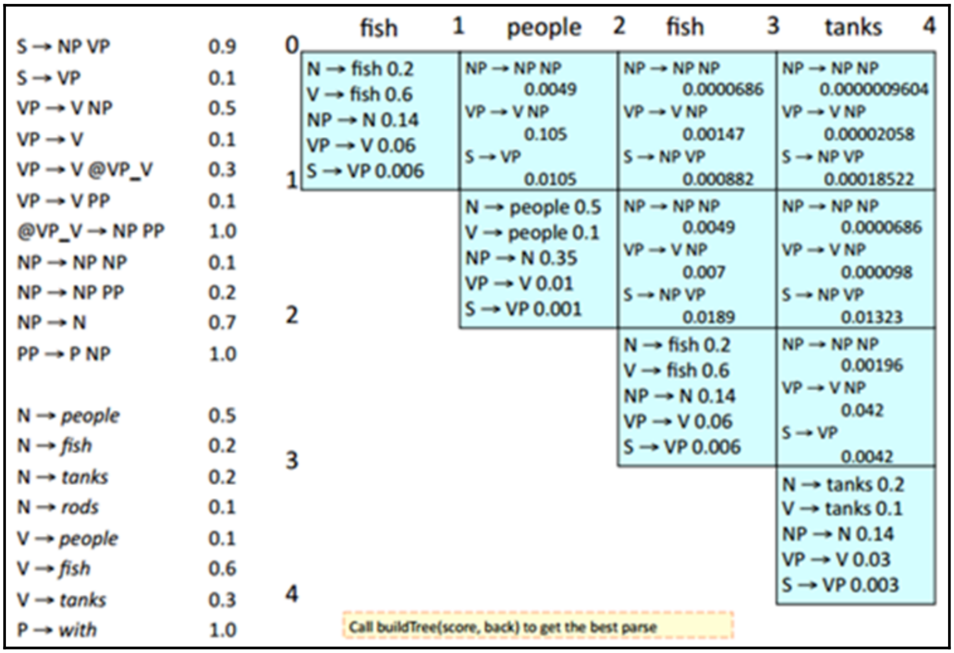
```
prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
if (prob > score[begin][end][A])
    score[begin]end][A] = prob
    back[begin][end][A] = new Triple(split,B,C)
```



S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish	people	fish	tanks
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105		
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

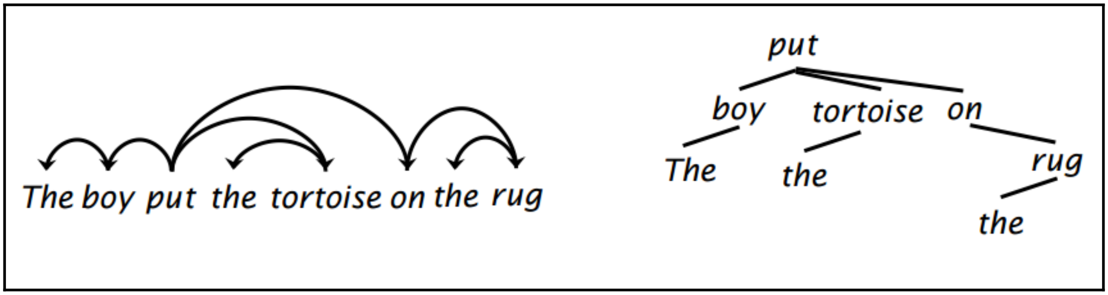
```
for split = begin+1 to end-1
  for A,B,C in nonterms
    prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
    if prob > score[begin][end][A]
      score[begin}end][A] = prob
      back[begin][end][A] = new Triple(split,B,C)
```

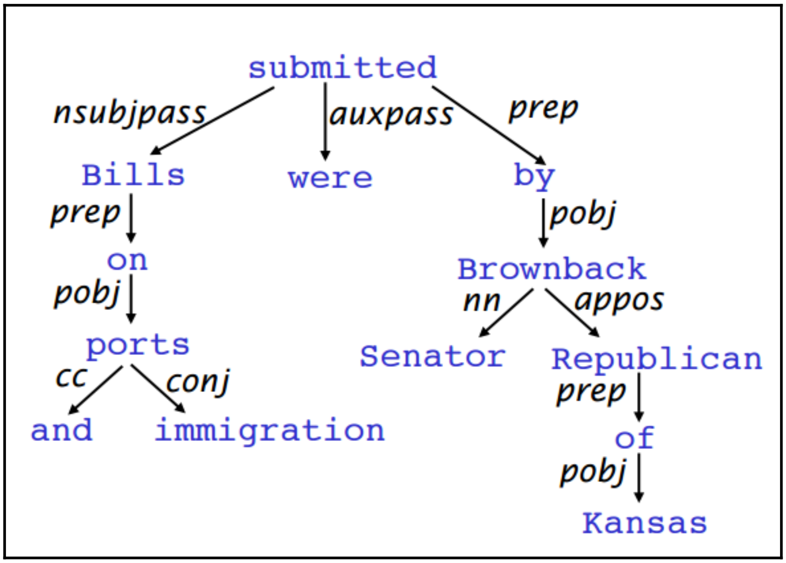


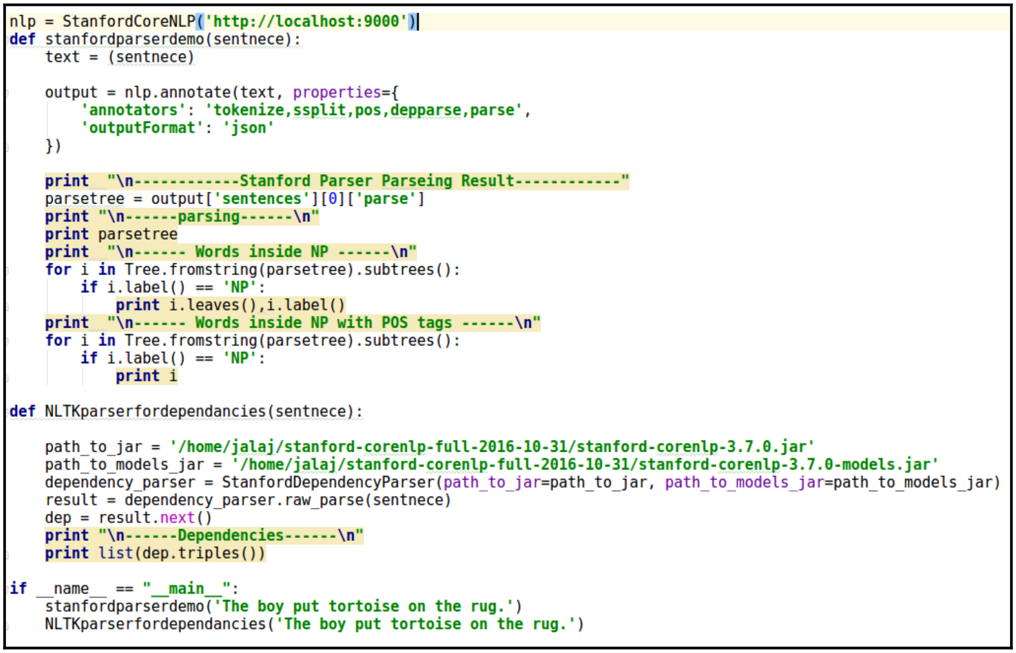
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish	people	fish	tanks
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

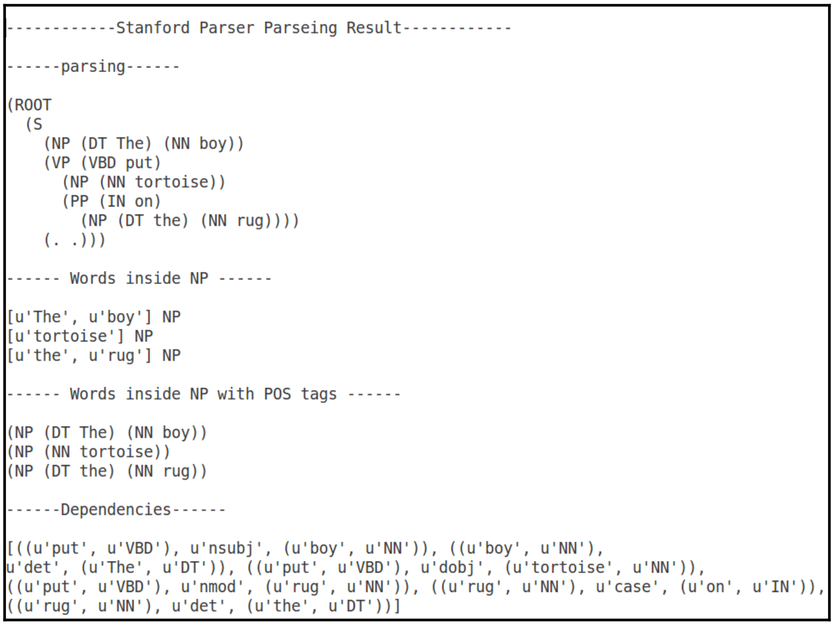
Call buildTree(score, back) to get the best parse


The boy put the tortoise on the rug
put
boy
tortoise
on
The
the
rug
the

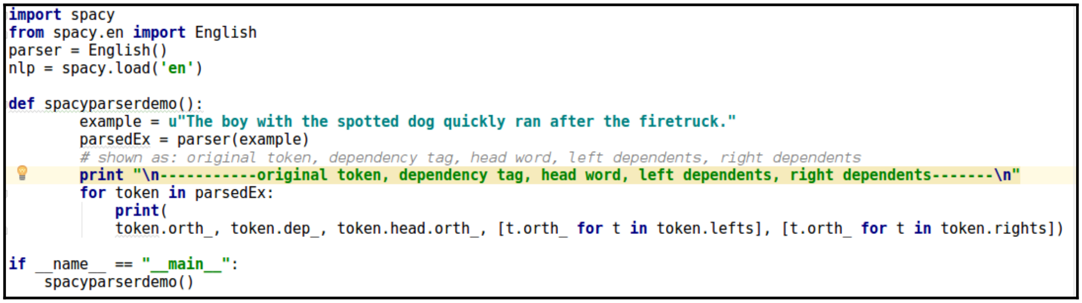

submitted
nsubjpass
auxpass
prep
Bills
were
by
prep
pobj
on
Brownback
pobj
nn
appos
ports
Senator
Republican
cc
conj
prep
and
immigration
of
pobj
Kansas



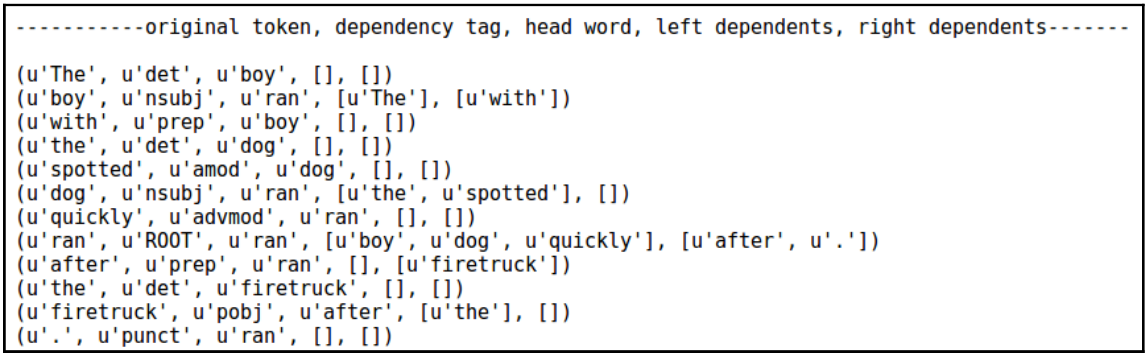
```python
nlp = StanfordCoreNLP('http://localhost:9000')
def stanfordparserdemo(sentnece):
    text = (sentnece)

    output = nlp.annotate(text, properties={
        'annotators': 'tokenize,ssplit,pos,depparse,parse',
        'outputFormat': 'json'
    })

    print  "\n------------Stanford Parser Parseing Result------------"
    parsetree = output['sentences'][0]['parse']
    print "\n------parsing------\n"
    print parsetree
    print  "\n------ Words inside NP ------\n"
    for i in Tree.fromstring(parsetree).subtrees():
        if i.label() == 'NP':
            print i.leaves(),i.label()
    print  "\n------ Words inside NP with POS tags ------\n"
    for i in Tree.fromstring(parsetree).subtrees():
        if i.label() == 'NP':
            print i

def NLTKparserfordependancies(sentnece):

    path_to_jar = '/home/jalaj/stanford-corenlp-full-2016-10-31/stanford-corenlp-3.7.0.jar'
    path_to_models_jar = '/home/jalaj/stanford-corenlp-full-2016-10-31/stanford-corenlp-3.7.0-models.jar'
    dependency_parser = StanfordDependencyParser(path_to_jar=path_to_jar, path_to_models_jar=path_to_models_jar)
    result = dependency_parser.raw_parse(sentnece)
    dep = result.next()
    print "\n------Dependencies------\n"
    print list(dep.triples())

if __name__ == "__main__":
    stanfordparserdemo('The boy put tortoise on the rug.')
    NLTKparserfordependancies('The boy put tortoise on the rug.')
```



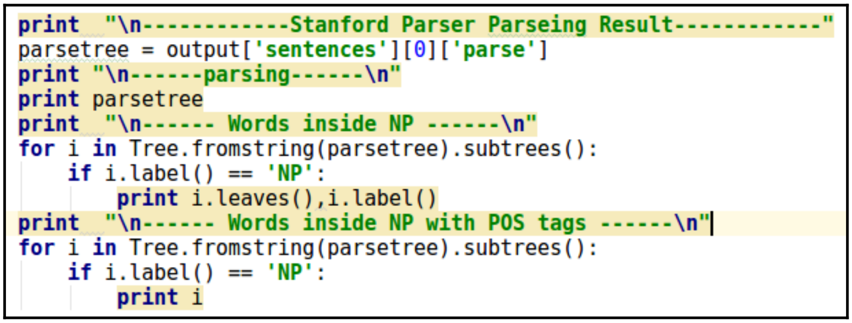
```
------------Stanford Parser Parseing Result------------

------parsing------

(ROOT
  (S
    (NP (DT The) (NN boy))
    (VP (VBD put)
      (NP (NN tortoise))
      (PP (IN on)
        (NP (DT the) (NN rug))))
    (. .)))

------ Words inside NP ------

[u'The', u'boy'] NP
[u'tortoise'] NP
[u'the', u'rug'] NP

------ Words inside NP with POS tags ------

(NP (DT The) (NN boy))
(NP (NN tortoise))
(NP (DT the) (NN rug))

------Dependencies------

[((u'put', u'VBD'), u'nsubj', (u'boy', u'NN')), ((u'boy', u'NN'),
u'det', (u'The', u'DT')), ((u'put', u'VBD'), u'dobj', (u'tortoise', u'NN')),
((u'put', u'VBD'), u'nmod', (u'rug', u'NN')), ((u'rug', u'NN'), u'case', (u'on', u'IN')),
((u'rug', u'NN'), u'det', (u'the', u'DT'))]
```



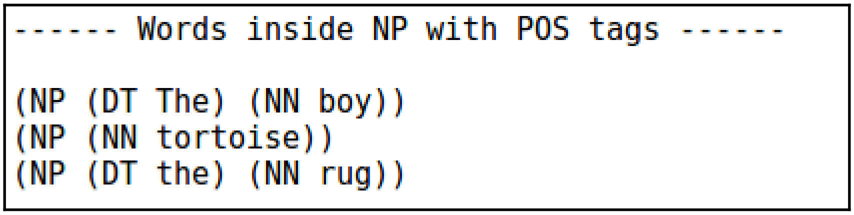
```
import spacy
from spacy.en import English
parser = English()
nlp = spacy.load('en')

def spacyparserdemo():
        example = u"The boy with the spotted dog quickly ran after the firetruck."
        parsedEx = parser(example)
        # shown as: original token, dependency tag, head word, left dependents, right dependents
        print "\n----------original token, dependency tag, head word, left dependents, right dependents-------\n"
        for token in parsedEx:
            print(
            token.orth_, token.dep_, token.head.orth_, [t.orth_ for t in token.lefts], [t.orth_ for t in token.rights])

if __name__ == "__main__":
    spacyparserdemo()
```



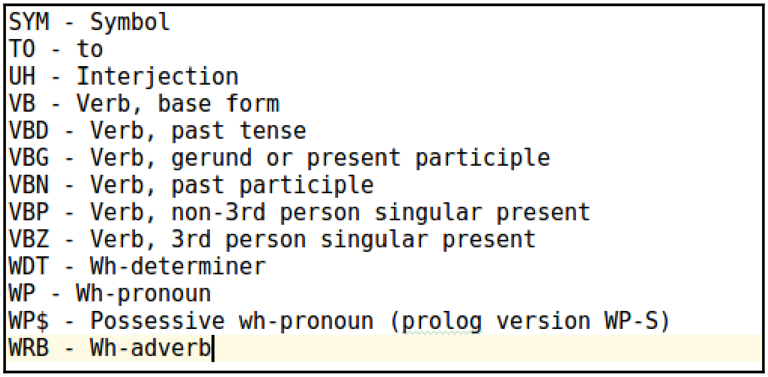
```
-----------original token, dependency tag, head word, left dependents, right dependents-------

(u'The', u'det', u'boy', [], [])
(u'boy', u'nsubj', u'ran', [u'The'], [u'with'])
(u'with', u'prep', u'boy', [], [])
(u'the', u'det', u'dog', [], [])
(u'spotted', u'amod', u'dog', [], [])
(u'dog', u'nsubj', u'ran', [u'the', u'spotted'], [])
(u'quickly', u'advmod', u'ran', [], [])
(u'ran', u'ROOT', u'ran', [u'boy', u'dog', u'quickly'], [u'after', u'.'])
(u'after', u'prep', u'ran', [], [u'firetruck'])
(u'the', u'det', u'firetruck', [], [])
(u'firetruck', u'pobj', u'after', [u'the'], [])
(u'.', u'punct', u'ran', [], [])
```



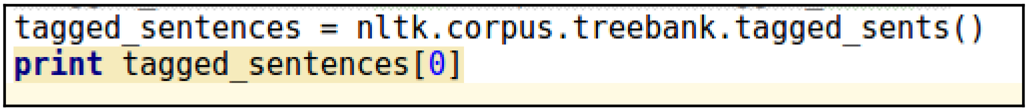
```
print  "\n------------Stanford Parser Parseing Result------------"
parsetree = output['sentences'][0]['parse']
print "\n------parsing------\n"
print parsetree
print  "\n------ Words inside NP ------\n"
for i in Tree.fromstring(parsetree).subtrees():
    if i.label() == 'NP':
        print i.leaves(),i.label()
print  "\n------ Words inside NP with POS tags ------\n"
for i in Tree.fromstring(parsetree).subtrees():
    if i.label() == 'NP':
        print i
```



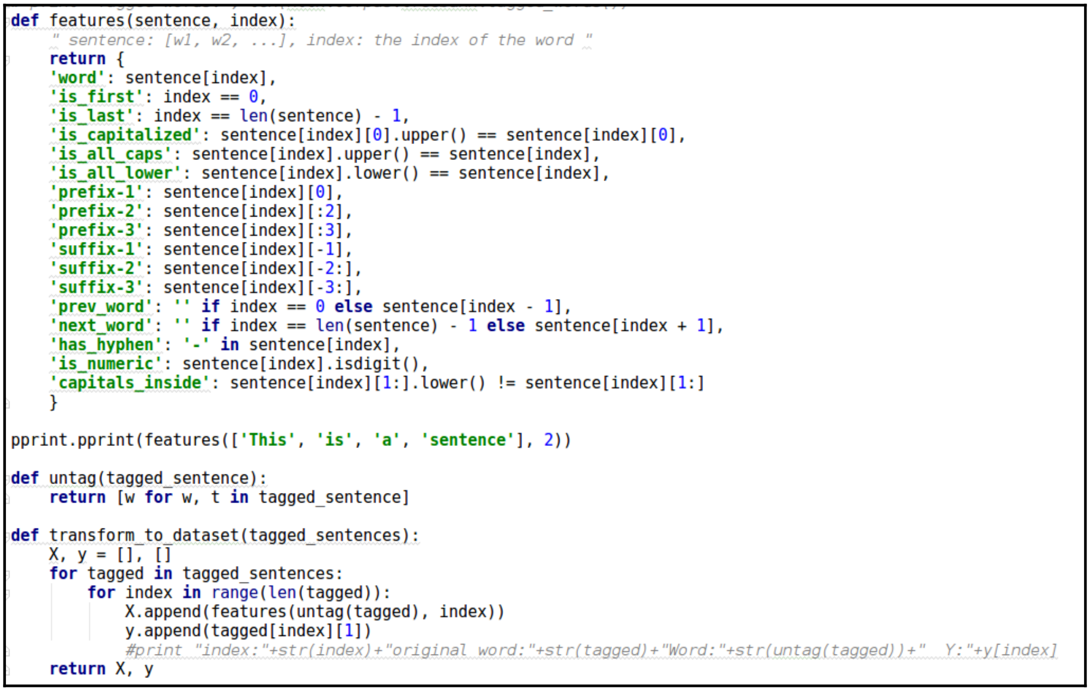
```
------ Words inside NP with POS tags ------

(NP (DT The) (NN boy))
(NP (NN tortoise))
(NP (DT the) (NN rug))
```



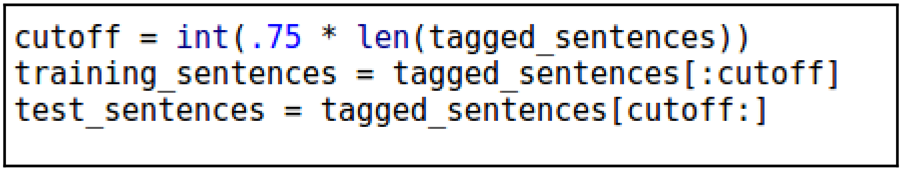
```
SYM - Symbol
TO - to
UH - Interjection
VB - Verb, base form
VBD - Verb, past tense
VBG - Verb, gerund or present participle
VBN - Verb, past participle
VBP - Verb, non-3rd person singular present
VBZ - Verb, 3rd person singular present
WDT - Wh-determiner
WP - Wh-pronoun
WP$ - Possessive wh-pronoun (prolog version WP-S)
WRB - Wh-adverb
```



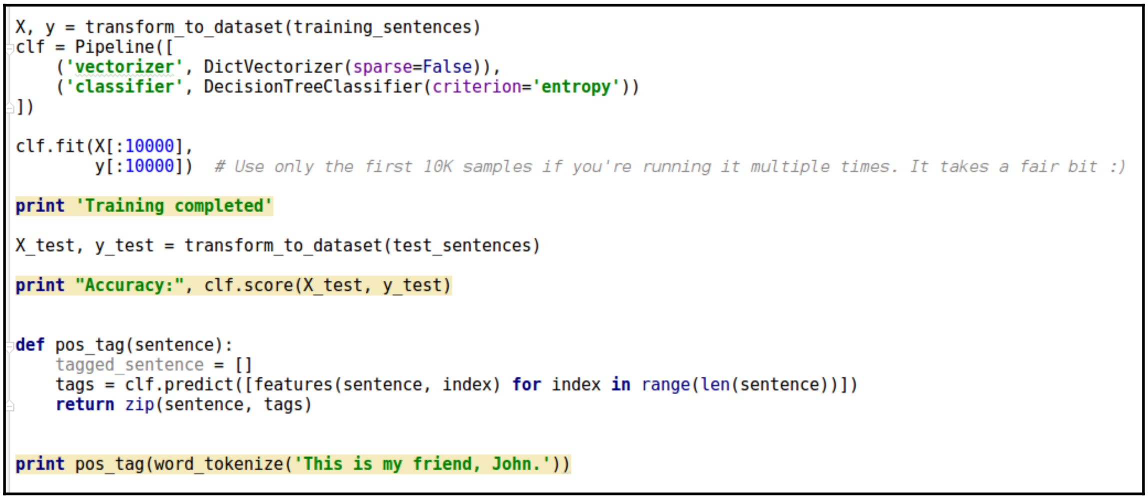
```
tagged_sentences = nltk.corpus.treebank.tagged_sents()
print tagged_sentences[0]
```



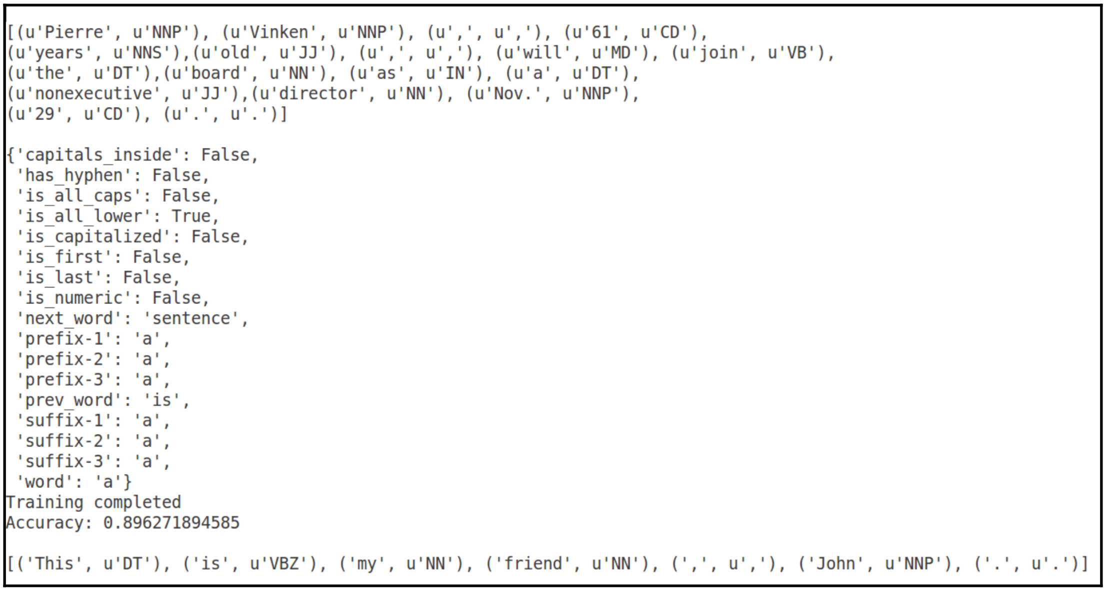
```
def features(sentence, index):
    " sentence: [w1, w2, ...], index: the index of the word "
    return {
    'word': sentence[index],
    'is_first': index == 0,
    'is_last': index == len(sentence) - 1,
    'is_capitalized': sentence[index][0].upper() == sentence[index][0],
    'is_all_caps': sentence[index].upper() == sentence[index],
    'is_all_lower': sentence[index].lower() == sentence[index],
    'prefix-1': sentence[index][0],
    'prefix-2': sentence[index][:2],
    'prefix-3': sentence[index][:3],
    'suffix-1': sentence[index][-1],
    'suffix-2': sentence[index][-2:],
    'suffix-3': sentence[index][-3:],
    'prev_word': '' if index == 0 else sentence[index - 1],
    'next_word': '' if index == len(sentence) - 1 else sentence[index + 1],
    'has_hyphen': '-' in sentence[index],
    'is_numeric': sentence[index].isdigit(),
    'capitals_inside': sentence[index][1:].lower() != sentence[index][1:]
    }

pprint.pprint(features(['This', 'is', 'a', 'sentence'], 2))

def untag(tagged_sentence):
    return [w for w, t in tagged_sentence]

def transform_to_dataset(tagged_sentences):
    X, y = [], []
    for tagged in tagged_sentences:
        for index in range(len(tagged)):
            X.append(features(untag(tagged), index))
            y.append(tagged[index][1])
            #print "index:"+str(index)+"original word:"+str(tagged)+"Word:"+str(untag(tagged))+"  Y:"+y[index]
    return X, y
```



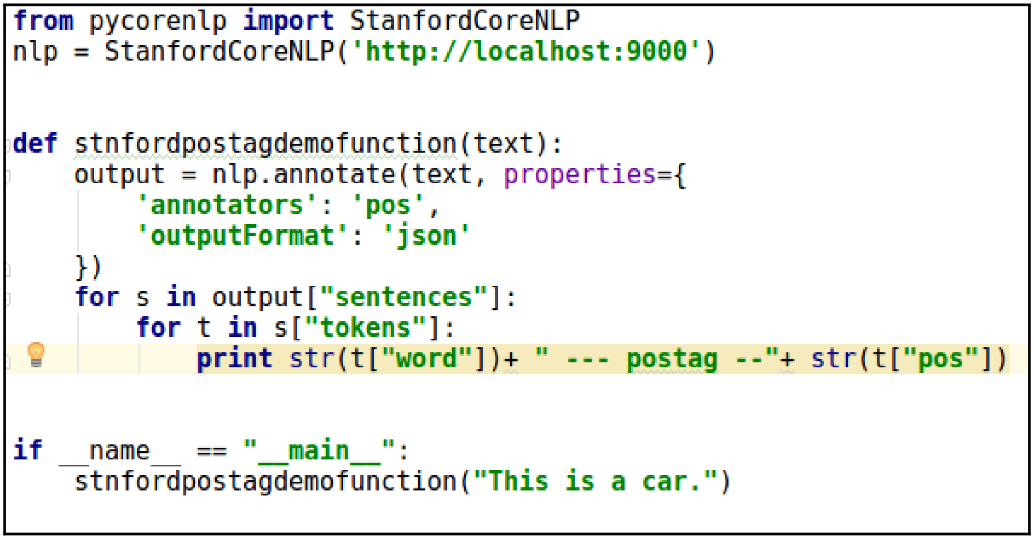
```
cutoff = int(.75 * len(tagged_sentences))
training_sentences = tagged_sentences[:cutoff]
test_sentences = tagged_sentences[cutoff:]
```



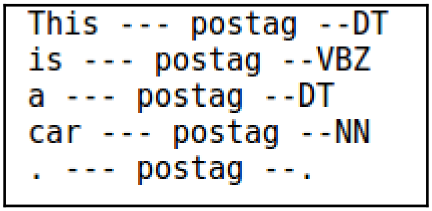
```
X, y = transform_to_dataset(training_sentences)
clf = Pipeline([
    ('vectorizer', DictVectorizer(sparse=False)),
    ('classifier', DecisionTreeClassifier(criterion='entropy'))
])

clf.fit(X[:10000],
        y[:10000])  # Use only the first 10K samples if you're running it multiple times. It takes a fair bit :)

print 'Training completed'

X_test, y_test = transform_to_dataset(test_sentences)

print "Accuracy:", clf.score(X_test, y_test)


def pos_tag(sentence):
    tagged_sentence = []
    tags = clf.predict([features(sentence, index) for index in range(len(sentence))])
    return zip(sentence, tags)


print pos_tag(word_tokenize('This is my friend, John.'))
```



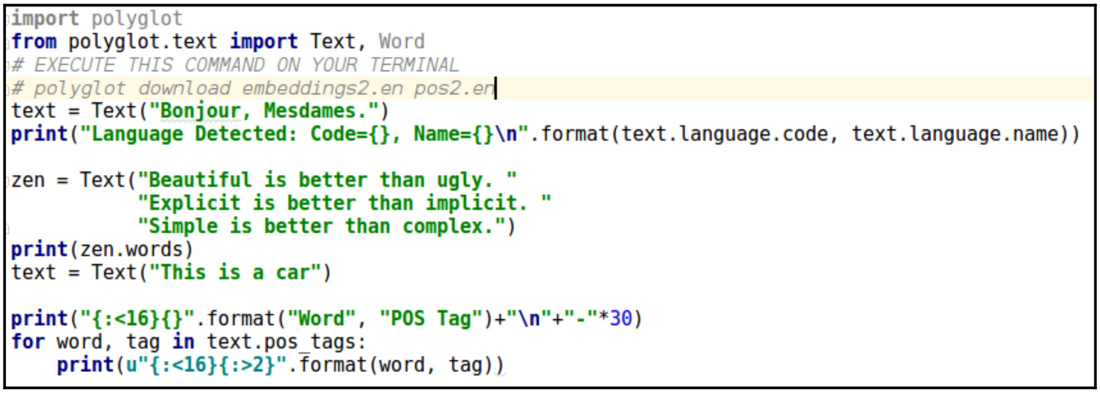
```
[(u'Pierre', u'NNP'), (u'Vinken', u'NNP'), (u',', u','), (u'61', u'CD'),
(u'years', u'NNS'),(u'old', u'JJ'), (u',', u','), (u'will', u'MD'), (u'join', u'VB'),
(u'the', u'DT'),(u'board', u'NN'), (u'as', u'IN'), (u'a', u'DT'),
(u'nonexecutive', u'JJ'),(u'director', u'NN'), (u'Nov.', u'NNP'),
(u'29', u'CD'), (u'.', u'.')]

{'capitals_inside': False,
 'has_hyphen': False,
 'is_all_caps': False,
 'is_all_lower': True,
 'is_capitalized': False,
 'is_first': False,
 'is_last': False,
 'is_numeric': False,
 'next_word': 'sentence',
 'prefix-1': 'a',
 'prefix-2': 'a',
 'prefix-3': 'a',
 'prev_word': 'is',
 'suffix-1': 'a',
 'suffix-2': 'a',
 'suffix-3': 'a',
 'word': 'a'}
Training completed
Accuracy: 0.896271894585

[('This', u'DT'), ('is', u'VBZ'), ('my', u'NN'), ('friend', u'NN'), (',', u','), ('John', u'NNP'), ('.', u'.')]
```



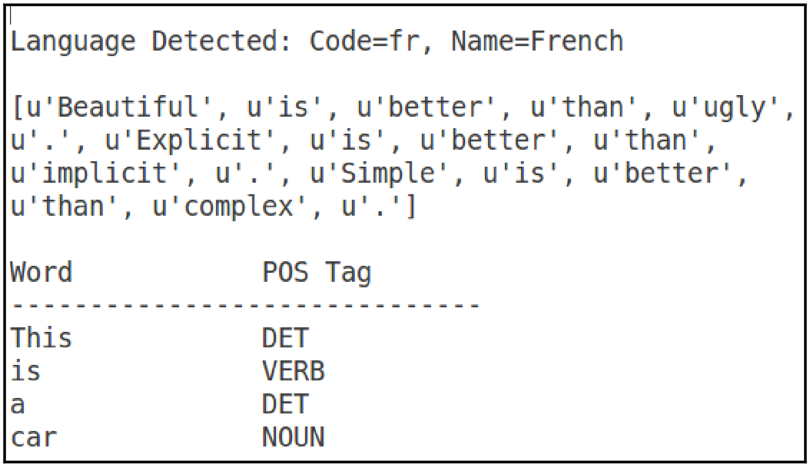
```
from pycorenlp import StanfordCoreNLP
nlp = StanfordCoreNLP('http://localhost:9000')


def stnfordpostagdemofunction(text):
    output = nlp.annotate(text, properties={
        'annotators': 'pos',
        'outputFormat': 'json'
    })
    for s in output["sentences"]:
        for t in s["tokens"]:
            print str(t["word"])+ " --- postag --"+ str(t["pos"])


if __name__ == "__main__":
    stnfordpostagdemofunction("This is a car.")
```



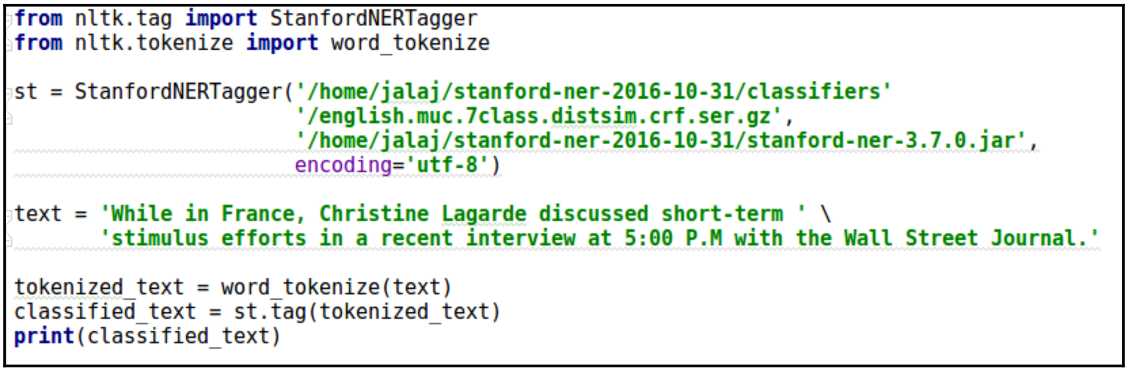
```
This --- postag --DT
is --- postag --VBZ
a --- postag --DT
car --- postag --NN
. --- postag --.
```



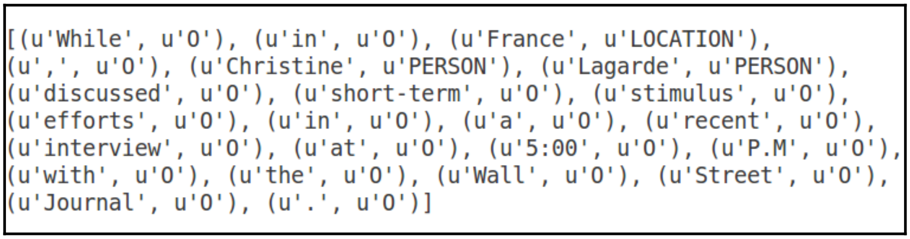
```
import polyglot
from polyglot.text import Text, Word
# EXECUTE THIS COMMAND ON YOUR TERMINAL
# polyglot download embeddings2.en pos2.en
text = Text("Bonjour, Mesdames.")
print("Language Detected: Code={}, Name={}\n".format(text.language.code, text.language.name))

zen = Text("Beautiful is better than ugly. "
           "Explicit is better than implicit. "
           "Simple is better than complex.")
print(zen.words)
text = Text("This is a car")

print("{:<16}{}".format("Word", "POS Tag")+"\n"+"-"*30)
for word, tag in text.pos_tags:
    print(u"{:<16}{:>2}".format(word, tag))
```



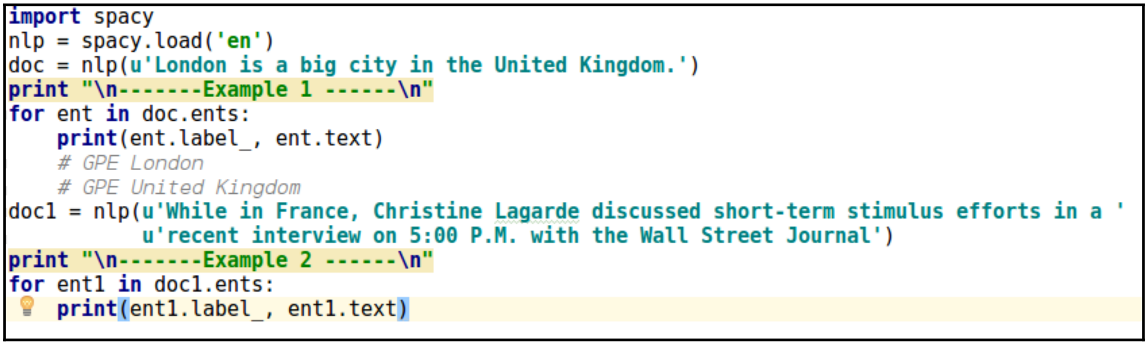
```
Language Detected: Code=fr, Name=French

[u'Beautiful', u'is', u'better', u'than', u'ugly',
u'.', u'Explicit', u'is', u'better', u'than',
u'implicit', u'.', u'Simple', u'is', u'better',
u'than', u'complex', u'.']

Word                POS Tag
------------------------------
This                DET
is                  VERB
a                   DET
car                 NOUN
```



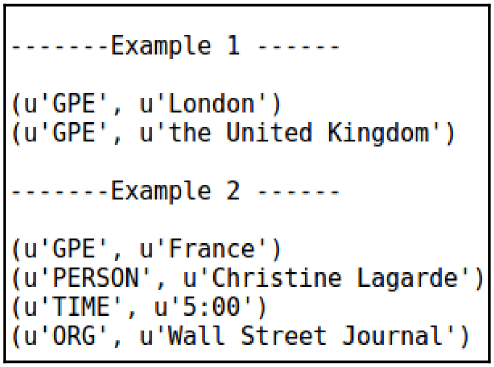
```python
from nltk.tag import StanfordNERTagger
from nltk.tokenize import word_tokenize

st = StanfordNERTagger('/home/jalaj/stanford-ner-2016-10-31/classifiers'
                       '/english.muc.7class.distsim.crf.ser.gz',
                       '/home/jalaj/stanford-ner-2016-10-31/stanford-ner-3.7.0.jar',
                       encoding='utf-8')

text = 'While in France, Christine Lagarde discussed short-term ' \
       'stimulus efforts in a recent interview at 5:00 P.M with the Wall Street Journal.'

tokenized_text = word_tokenize(text)
classified_text = st.tag(tokenized_text)
print(classified_text)
```



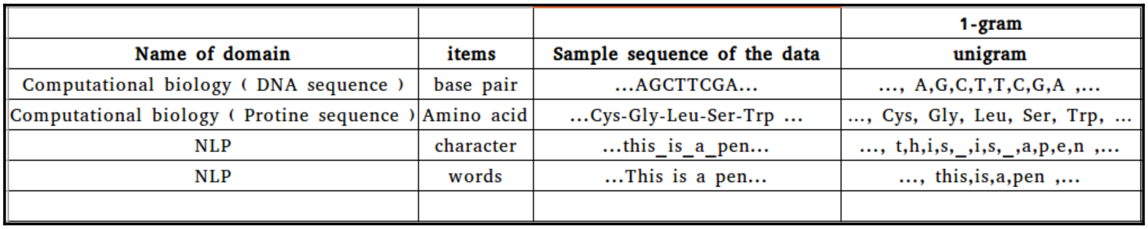
```
[(u'While', u'O'), (u'in', u'O'), (u'France', u'LOCATION'),
(u',', u'O'), (u'Christine', u'PERSON'), (u'Lagarde', u'PERSON'),
(u'discussed', u'O'), (u'short-term', u'O'), (u'stimulus', u'O'),
(u'efforts', u'O'), (u'in', u'O'), (u'a', u'O'), (u'recent', u'O'),
(u'interview', u'O'), (u'at', u'O'), (u'5:00', u'O'), (u'P.M', u'O'),
(u'with', u'O'), (u'the', u'O'), (u'Wall', u'O'), (u'Street', u'O'),
(u'Journal', u'O'), (u'.', u'O')]
```



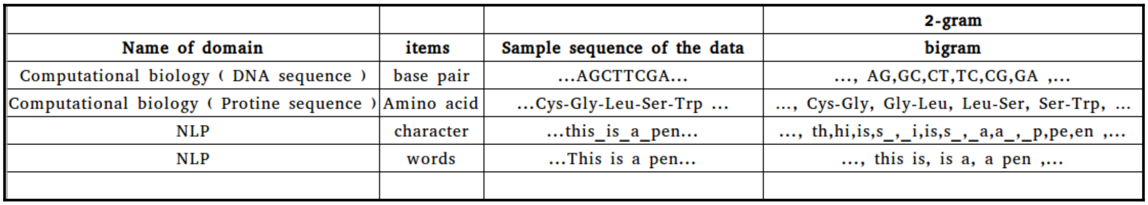
```
import spacy
nlp = spacy.load('en')
doc = nlp(u'London is a big city in the United Kingdom.')
print "\n-------Example 1 ------\n"
for ent in doc.ents:
    print(ent.label_, ent.text)
    # GPE London
    # GPE United Kingdom
doc1 = nlp(u'While in France, Christine Lagarde discussed short-term stimulus efforts in a '
           u'recent interview on 5:00 P.M. with the Wall Street Journal')
print "\n-------Example 2 ------\n"
for ent1 in doc1.ents:
    print(ent1.label_, ent1.text)
```



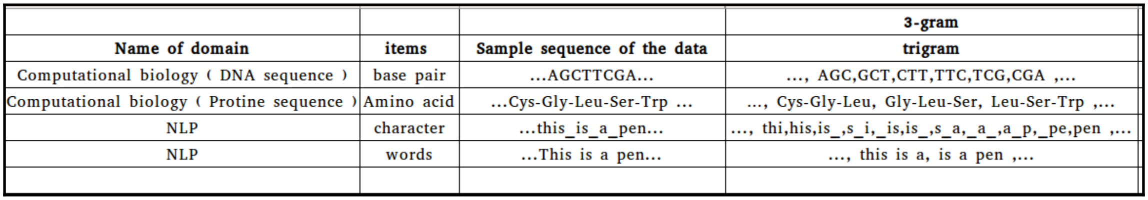
```
-------Example 1 ------

(u'GPE', u'London')
(u'GPE', u'the United Kingdom')

-------Example 2 ------

(u'GPE', u'France')
(u'PERSON', u'Christine Lagarde')
(u'TIME', u'5:00')
(u'ORG', u'Wall Street Journal')
```



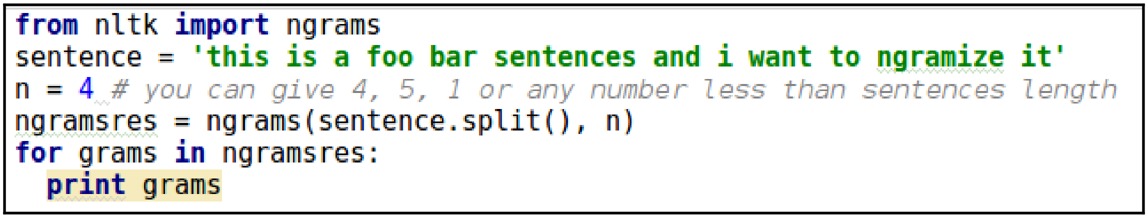
			1-gram
Name of domain	**items**	**Sample sequence of the data**	**unigram**
Computational biology (DNA sequence)	base pair	...AGCTTCGA...	..., A,G,C,T,T,C,G,A ,...
Computational biology (Protine sequence)	Amino acid	...Cys-Gly-Leu-Ser-Trp ...	..., Cys, Gly, Leu, Ser, Trp, ...
NLP	character	...this_is_a_pen...	..., t,h,i,s,_,i,s,_,a,p,e,n ,...
NLP	words	...This is a pen...	..., this,is,a,pen ,...



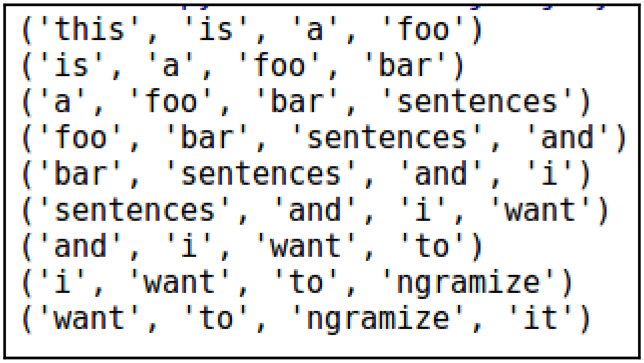
			2-gram
Name of domain	**items**	**Sample sequence of the data**	**bigram**
Computational biology (DNA sequence)	base pair	...AGCTTCGA...	..., AG,GC,CT,TC,CG,GA ,...
Computational biology (Protine sequence)	Amino acid	...Cys-Gly-Leu-Ser-Trp ...	..., Cys-Gly, Gly-Leu, Leu-Ser, Ser-Trp, ...
NLP	character	...this_is_a_pen...	..., th,hi,is,s_,_i,is,s_,_a,a_,_p,pe,en ,...
NLP	words	...This is a pen...	..., this is, is a, a pen ,...



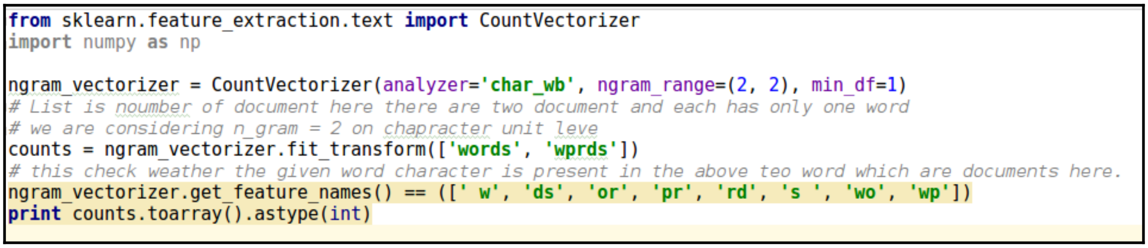
			3-gram
Name of domain	items	Sample sequence of the data	trigram
Computational biology (DNA sequence)	base pair	...AGCTTCGA...	..., AGC,GCT,CTT,TTC,TCG,CGA ,...
Computational biology (Protine sequence)	Amino acid	...Cys-Gly-Leu-Ser-Trp ...	..., Cys-Gly-Leu, Gly-Leu-Ser, Leu-Ser-Trp ,...
NLP	character	...this_is_a_pen...	..., thi,his,is_,s_i,_is,is_,s_a,_a_,a_p,_pe,pen ,...
NLP	words	...This is a pen...	..., this is a, is a pen ,...



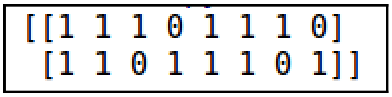
```
from nltk import ngrams
sentence = 'this is a foo bar sentences and i want to ngramize it'
n = 4 # you can give 4, 5, 1 or any number less than sentences length
ngramsres = ngrams(sentence.split(), n)
for grams in ngramsres:
  print grams
```



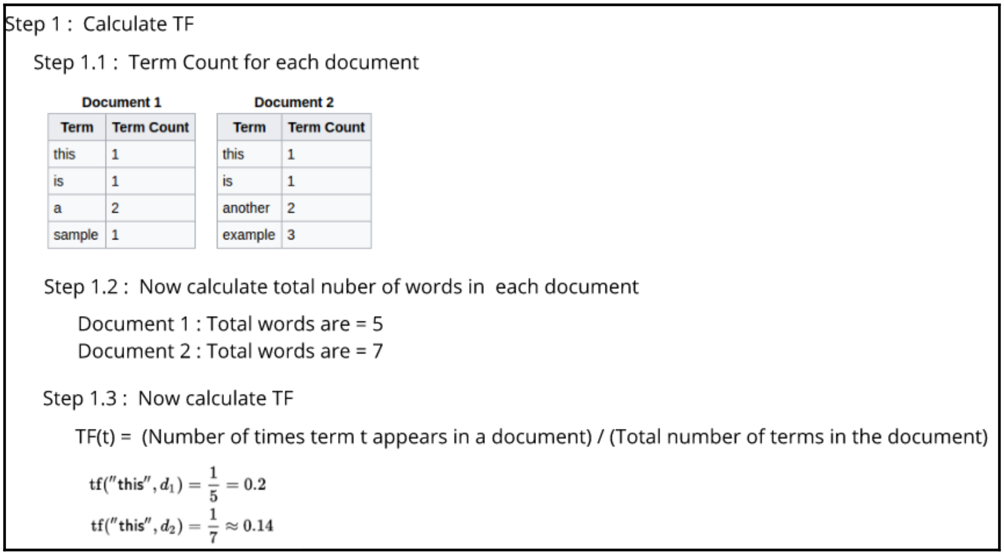
```
('this', 'is', 'a', 'foo')
('is', 'a', 'foo', 'bar')
('a', 'foo', 'bar', 'sentences')
('foo', 'bar', 'sentences', 'and')
('bar', 'sentences', 'and', 'i')
('sentences', 'and', 'i', 'want')
('and', 'i', 'want', 'to')
('i', 'want', 'to', 'ngramize')
('want', 'to', 'ngramize', 'it')
```



```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np

ngram_vectorizer = CountVectorizer(analyzer='char_wb', ngram_range=(2, 2), min_df=1)
# List is noumber of document here there are two document and each has only one word
# we are considering n_gram = 2 on chapracter unit leve
counts = ngram_vectorizer.fit_transform(['words', 'wprds'])
# this check weather the given word character is present in the above teo word which are documents here.
ngram_vectorizer.get_feature_names() == ([' w', 'ds', 'or', 'pr', 'rd', 's ', 'wo', 'wp'])
print counts.toarray().astype(int)
```



```
[[1 1 1 0 1 1 1 0]
 [1 1 0 1 1 1 0 1]]
```



Step 1 : Calculate TF

Step 1.1 : Term Count for each document

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

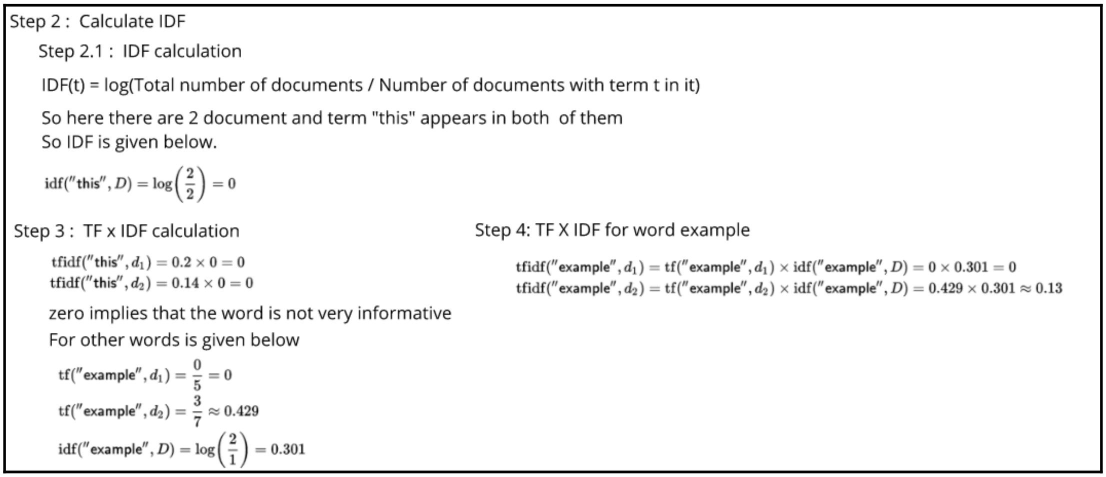
Step 1.2 : Now calculate total nuber of words in each document

Document 1 : Total words are = 5
Document 2 : Total words are = 7

Step 1.3 : Now calculate TF

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

$$\mathrm{tf}(''\mathsf{this}'', d_1) = \frac{1}{5} = 0.2$$
$$\mathrm{tf}(''\mathsf{this}'', d_2) = \frac{1}{7} \approx 0.14$$



Step 2 : Calculate IDF

Step 2.1 : IDF calculation

IDF(t) = log(Total number of documents / Number of documents with term t in it)

So here there are 2 document and term "this" appears in both of them
So IDF is given below.

$$\mathrm{idf}(''\mathsf{this}'', D) = \log\left(\frac{2}{2}\right) = 0$$

Step 3 : TF x IDF calculation

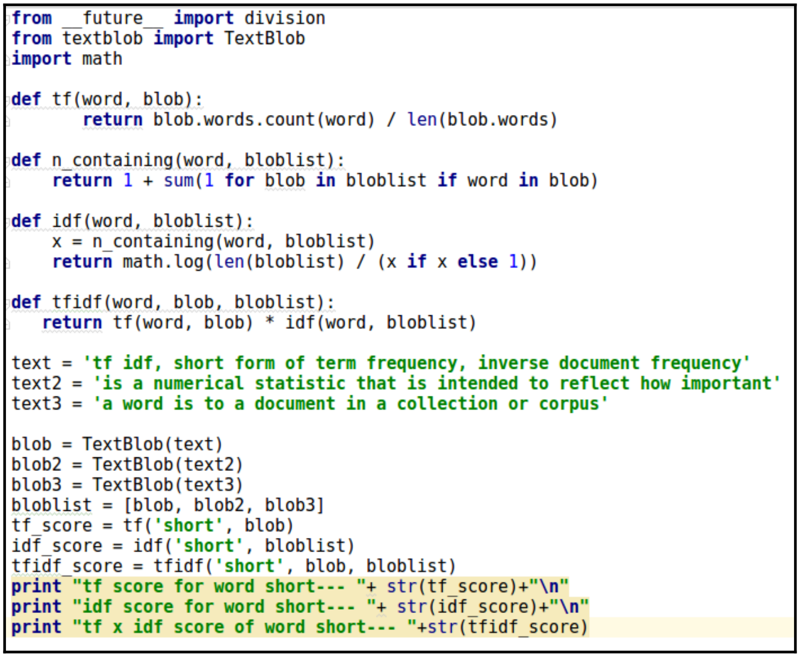
$$\mathrm{tfidf}(''\mathsf{this}'', d_1) = 0.2 \times 0 = 0$$
$$\mathrm{tfidf}(''\mathsf{this}'', d_2) = 0.14 \times 0 = 0$$

zero implies that the word is not very informative
For other words is given below

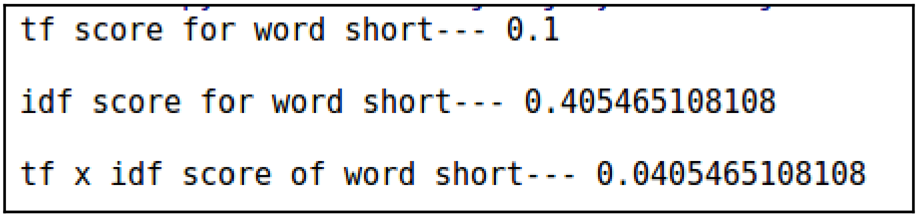
$$\mathrm{tf}(''\mathsf{example}'', d_1) = \frac{0}{5} = 0$$
$$\mathrm{tf}(''\mathsf{example}'', d_2) = \frac{3}{7} \approx 0.429$$
$$\mathrm{idf}(''\mathsf{example}'', D) = \log\left(\frac{2}{1}\right) = 0.301$$

Step 4: TF X IDF for word example

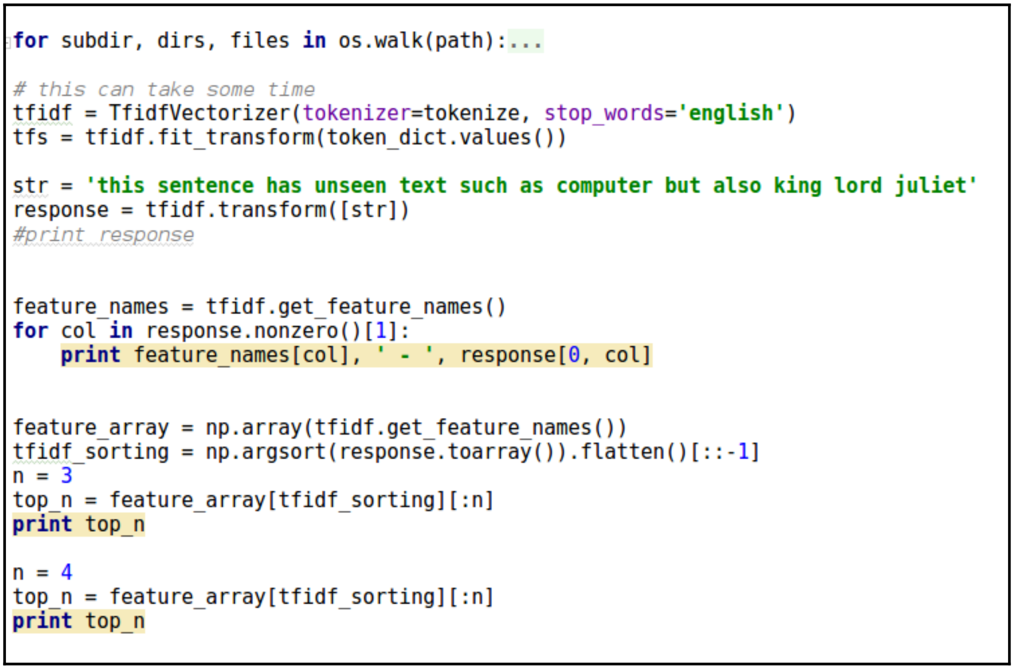
$$\mathrm{tfidf}(''\mathsf{example}'', d_1) = \mathrm{tf}(''\mathsf{example}'', d_1) \times \mathrm{idf}(''\mathsf{example}'', D) = 0 \times 0.301 = 0$$
$$\mathrm{tfidf}(''\mathsf{example}'', d_2) = \mathrm{tf}(''\mathsf{example}'', d_2) \times \mathrm{idf}(''\mathsf{example}'', D) = 0.429 \times 0.301 \approx 0.13$$



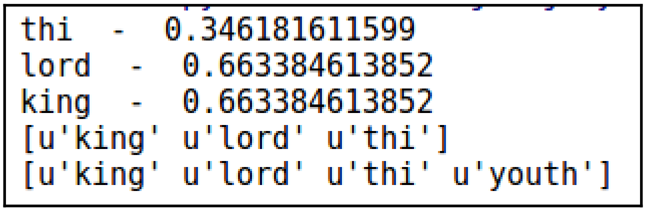
```
from __future__ import division
from textblob import TextBlob
import math

def tf(word, blob):
       return blob.words.count(word) / len(blob.words)

def n_containing(word, bloblist):
    return 1 + sum(1 for blob in bloblist if word in blob)

def idf(word, bloblist):
    x = n_containing(word, bloblist)
    return math.log(len(bloblist) / (x if x else 1))

def tfidf(word, blob, bloblist):
   return tf(word, blob) * idf(word, bloblist)

text = 'tf idf, short form of term frequency, inverse document frequency'
text2 = 'is a numerical statistic that is intended to reflect how important'
text3 = 'a word is to a document in a collection or corpus'

blob = TextBlob(text)
blob2 = TextBlob(text2)
blob3 = TextBlob(text3)
bloblist = [blob, blob2, blob3]
tf_score = tf('short', blob)
idf_score = idf('short', bloblist)
tfidf_score = tfidf('short', blob, bloblist)
print "tf score for word short--- "+ str(tf_score)+"\n"
print "idf score for word short--- "+ str(idf_score)+"\n"
print "tf x idf score of word short--- "+str(tfidf_score)
```



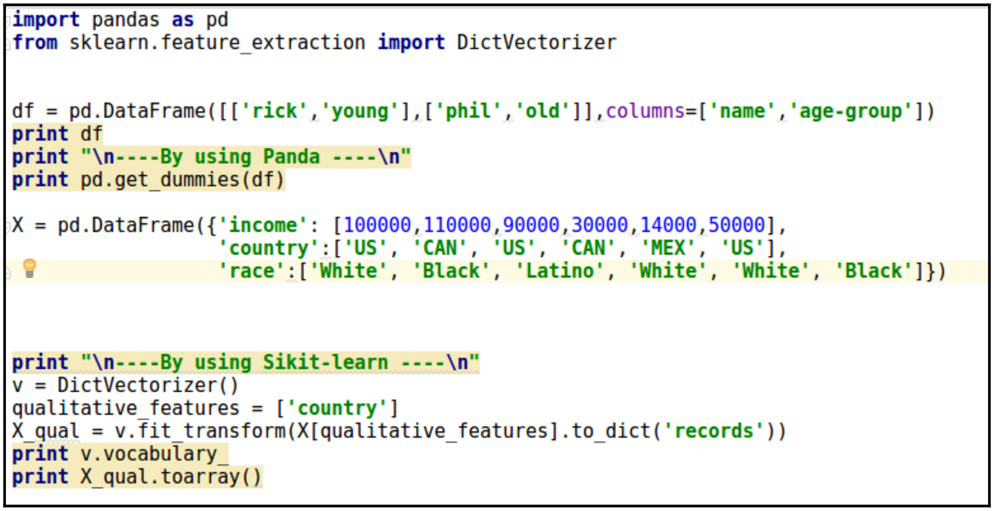
```
tf score for word short--- 0.1

idf score for word short--- 0.405465108108

tf x idf score of word short--- 0.0405465108108
```



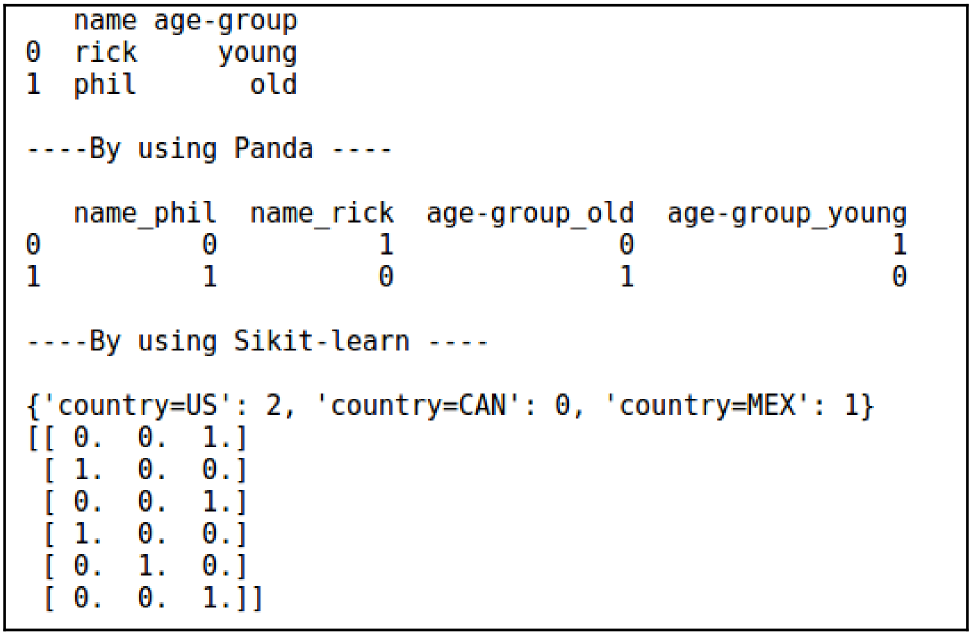
```
for subdir, dirs, files in os.walk(path):...

# this can take some time
tfidf = TfidfVectorizer(tokenizer=tokenize, stop_words='english')
tfs = tfidf.fit_transform(token_dict.values())

str = 'this sentence has unseen text such as computer but also king lord juliet'
response = tfidf.transform([str])
#print response


feature_names = tfidf.get_feature_names()
for col in response.nonzero()[1]:
    print feature_names[col], ' - ', response[0, col]


feature_array = np.array(tfidf.get_feature_names())
tfidf_sorting = np.argsort(response.toarray()).flatten()[::-1]
n = 3
top_n = feature_array[tfidf_sorting][:n]
print top_n

n = 4
top_n = feature_array[tfidf_sorting][:n]
print top_n
```



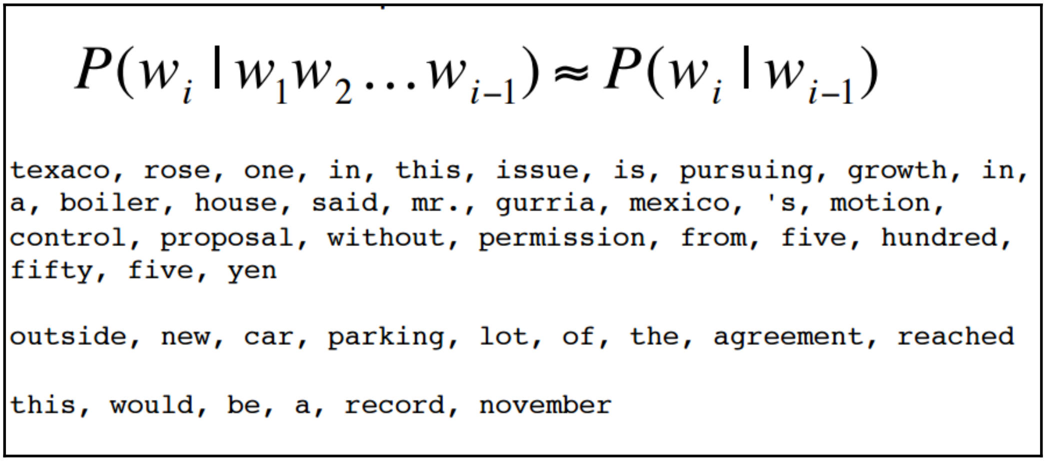
```
thi  -  0.346181611599
lord  -  0.663384613852
king  -  0.663384613852
[u'king' u'lord' u'thi']
[u'king' u'lord' u'thi' u'youth']
```



```
import pandas as pd
from sklearn.feature_extraction import DictVectorizer


df = pd.DataFrame([['rick','young'],['phil','old']],columns=['name','age-group'])
print df
print "\n----By using Panda ----\n"
print pd.get_dummies(df)

X = pd.DataFrame({'income': [100000,110000,90000,30000,14000,50000],
                  'country':['US', 'CAN', 'US', 'CAN', 'MEX', 'US'],
                  'race':['White', 'Black', 'Latino', 'White', 'White', 'Black']})



print "\n----By using Sikit-learn ----\n"
v = DictVectorizer()
qualitative_features = ['country']
X_qual = v.fit_transform(X[qualitative_features].to_dict('records'))
print v.vocabulary_
print X_qual.toarray()
```

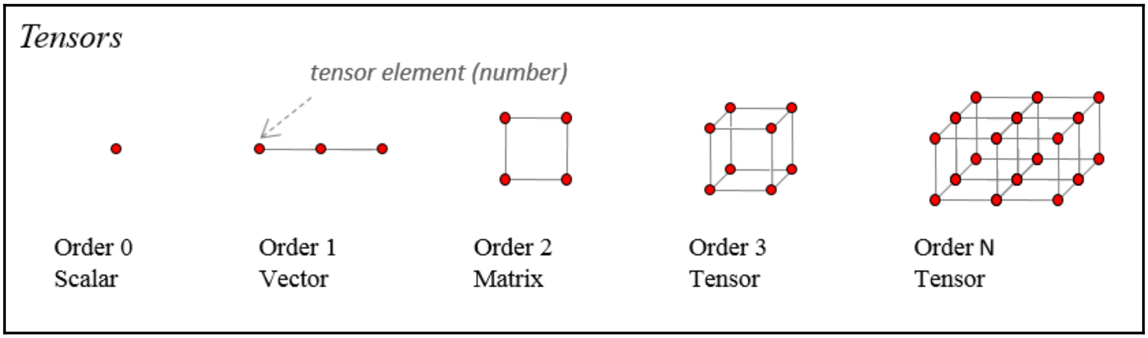


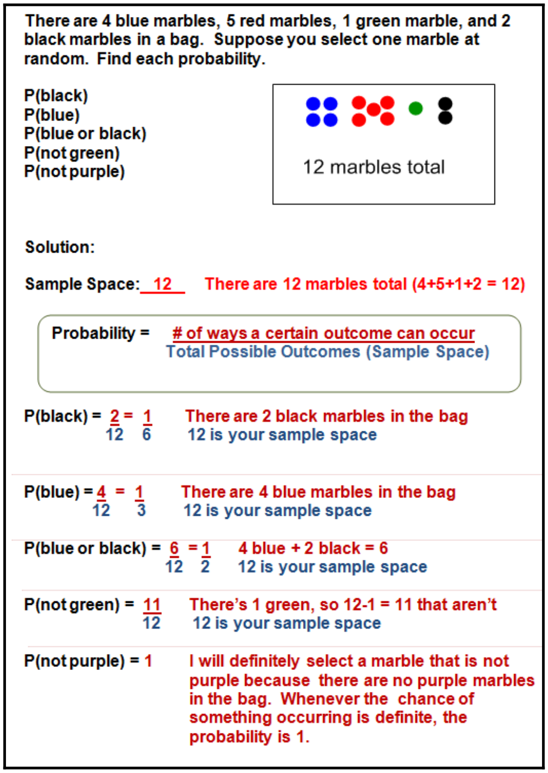
```
   name age-group
0  rick     young
1  phil       old

----By using Panda ----

   name_phil  name_rick  age-group_old  age-group_young
0          0          1              0                1
1          1          0              1                0

----By using Sikit-learn ----

{'country=US': 2, 'country=CAN': 0, 'country=MEX': 1}
[[ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```



$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

```
texaco, rose, one, in, this, issue, is, pursuing, growth, in,
a, boiler, house, said, mr., gurria, mexico, 's, motion,
control, proposal, without, permission, from, five, hundred,
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november
```





There are 4 blue marbles, 5 red marbles, 1 green marble, and 2 black marbles in a bag. Suppose you select one marble at random. Find each probability.

P(black)
P(blue)
P(blue or black)
P(not green)
P(not purple)

●● ●●● ● ●
●● ●●   ●

12 marbles total

Solution:

Sample Space: 12   There are 12 marbles total (4+5+1+2 = 12)

Probability = $\frac{\text{\# of ways a certain outcome can occur}}{\text{Total Possible Outcomes (Sample Space)}}$

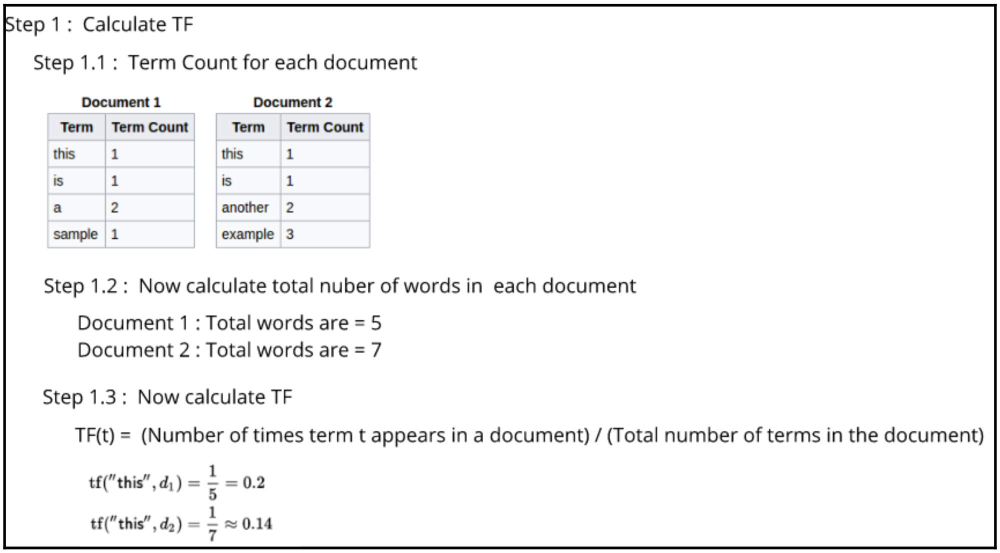
P(black) = $\frac{2}{12} = \frac{1}{6}$ There are 2 black marbles in the bag
12 is your sample space

P(blue) = $\frac{4}{12} = \frac{1}{3}$ There are 4 blue marbles in the bag
12 is your sample space

P(blue or black) = $\frac{6}{12} = \frac{1}{2}$ 4 blue + 2 black = 6
12 is your sample space

P(not green) = $\frac{11}{12}$ There's 1 green, so 12-1 = 11 that aren't
12 is your sample space

P(not purple) = 1   I will definitely select a marble that is not purple because there are no purple marbles in the bag. Whenever the chance of something occurring is definite, the probability is 1.



Step 1 : Calculate TF

Step 1.1 : Term Count for each document

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Step 1.2 : Now calculate total nuber of words in each document

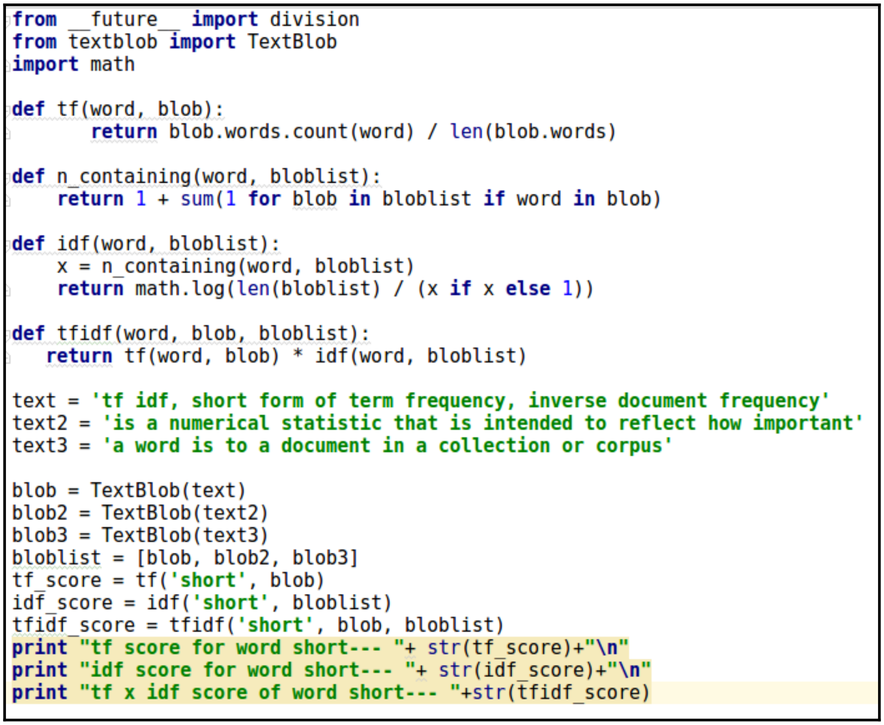
Document 1 : Total words are = 5
Document 2 : Total words are = 7

Step 1.3 : Now calculate TF

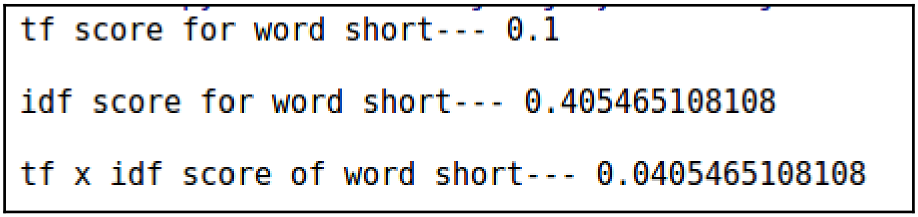
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

$$\mathrm{tf}(''\mathrm{this}'', d_1) = \frac{1}{5} = 0.2$$

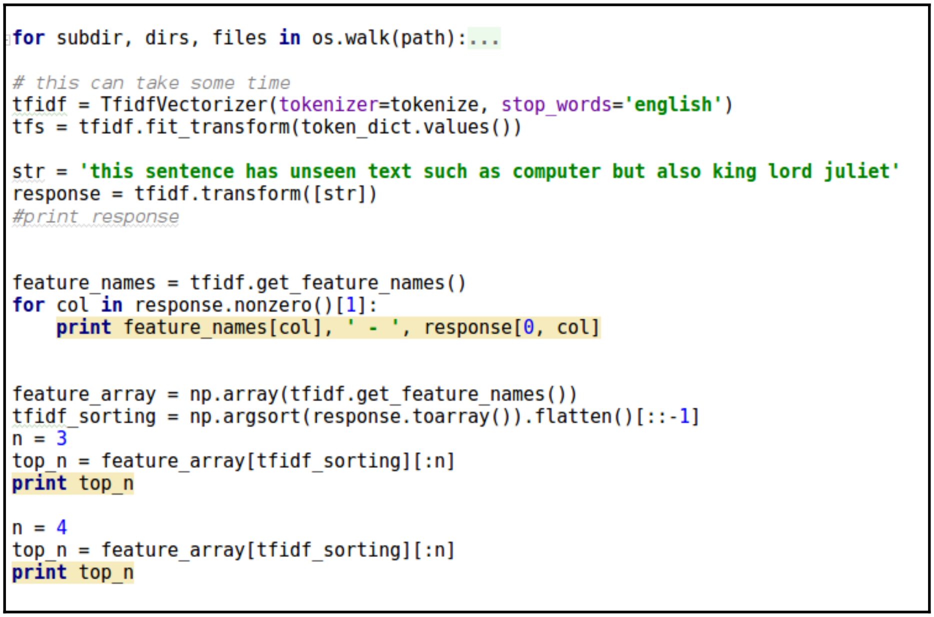
$$\mathrm{tf}(''\mathrm{this}'', d_2) = \frac{1}{7} \approx 0.14$$



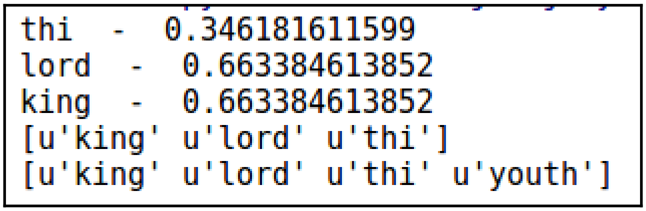
```
from __future__ import division
from textblob import TextBlob
import math

def tf(word, blob):
       return blob.words.count(word) / len(blob.words)

def n_containing(word, bloblist):
    return 1 + sum(1 for blob in bloblist if word in blob)

def idf(word, bloblist):
    x = n_containing(word, bloblist)
    return math.log(len(bloblist) / (x if x else 1))

def tfidf(word, blob, bloblist):
   return tf(word, blob) * idf(word, bloblist)

text = 'tf idf, short form of term frequency, inverse document frequency'
text2 = 'is a numerical statistic that is intended to reflect how important'
text3 = 'a word is to a document in a collection or corpus'

blob = TextBlob(text)
blob2 = TextBlob(text2)
blob3 = TextBlob(text3)
bloblist = [blob, blob2, blob3]
tf_score = tf('short', blob)
idf_score = idf('short', bloblist)
tfidf_score = tfidf('short', blob, bloblist)
print "tf score for word short--- "+ str(tf_score)+"\n"
print "idf score for word short--- "+ str(idf_score)+"\n"
print "tf x idf score of word short--- "+str(tfidf_score)
```



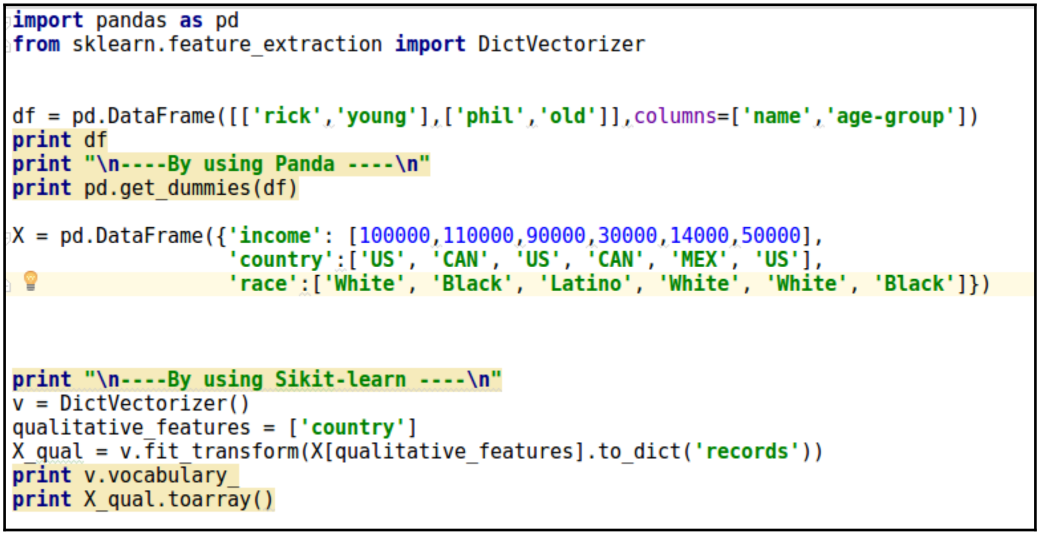
```
tf score for word short--- 0.1

idf score for word short--- 0.405465108108

tf x idf score of word short--- 0.0405465108108
```



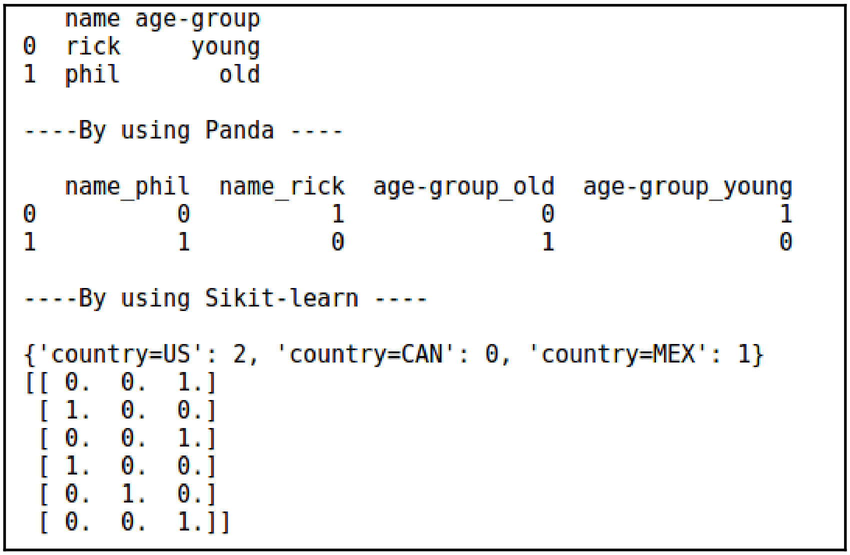
```
for subdir, dirs, files in os.walk(path):...

# this can take some time
tfidf = TfidfVectorizer(tokenizer=tokenize, stop_words='english')
tfs = tfidf.fit_transform(token_dict.values())

str = 'this sentence has unseen text such as computer but also king lord juliet'
response = tfidf.transform([str])
#print response


feature_names = tfidf.get_feature_names()
for col in response.nonzero()[1]:
    print feature_names[col], ' - ', response[0, col]


feature_array = np.array(tfidf.get_feature_names())
tfidf_sorting = np.argsort(response.toarray()).flatten()[::-1]
n = 3
top_n = feature_array[tfidf_sorting][:n]
print top_n

n = 4
top_n = feature_array[tfidf_sorting][:n]
print top_n
```



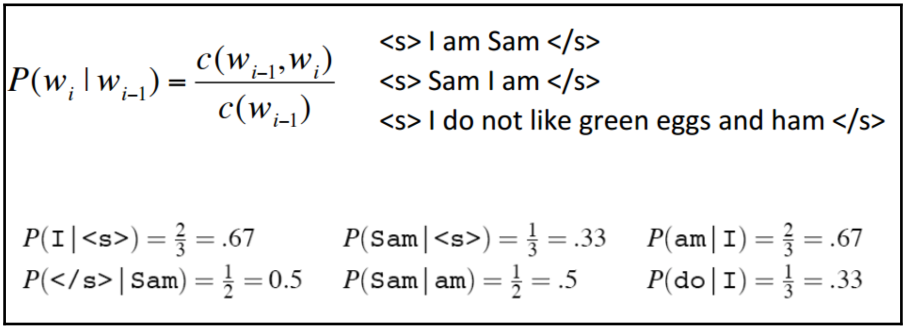
```
thi   -  0.346181611599
lord  -  0.663384613852
king  -  0.663384613852
[u'king' u'lord' u'thi']
[u'king' u'lord' u'thi' u'youth']
```



```
import pandas as pd
from sklearn.feature_extraction import DictVectorizer


df = pd.DataFrame([['rick','young'],['phil','old']],columns=['name','age-group'])
print df
print "\n----By using Panda ----\n"
print pd.get_dummies(df)

X = pd.DataFrame({'income': [100000,110000,90000,30000,14000,50000],
                  'country':['US', 'CAN', 'US', 'CAN', 'MEX', 'US'],
                  'race':['White', 'Black', 'Latino', 'White', 'White', 'Black']})



print "\n----By using Sikit-learn ----\n"
v = DictVectorizer()
qualitative_features = ['country']
X_qual = v.fit_transform(X[qualitative_features].to_dict('records'))
print v.vocabulary_
print X_qual.toarray()
```



```
   name age-group
0  rick     young
1  phil       old

----By using Panda ----

   name_phil  name_rick  age-group_old  age-group_young
0          0          1              0                1
1          1          0              1                0

----By using Sikit-learn ----

{'country=US': 2, 'country=CAN': 0, 'country=MEX': 1}
[[ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```



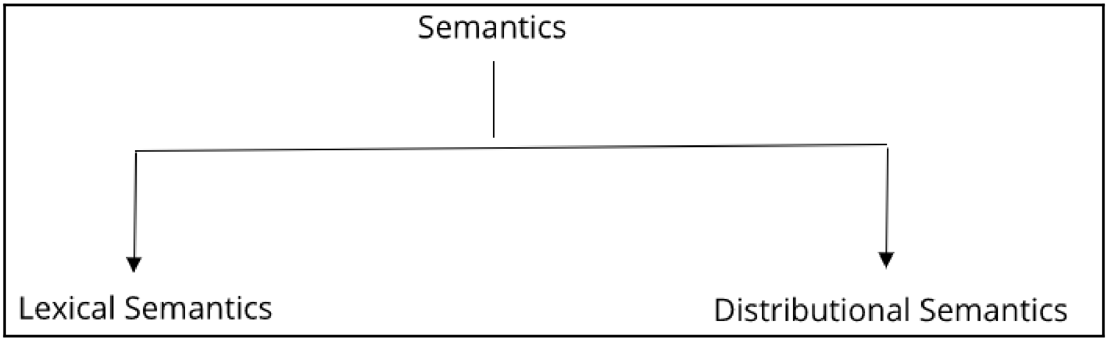
$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

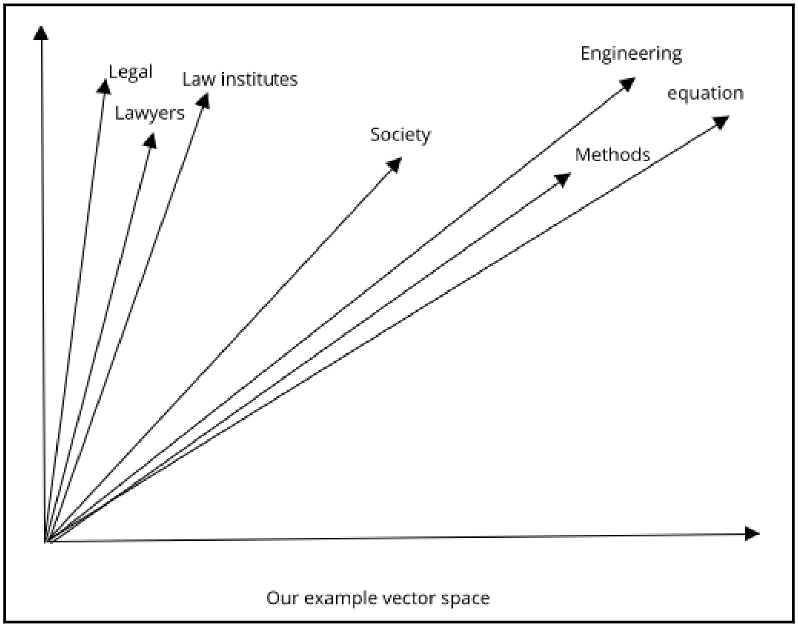
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

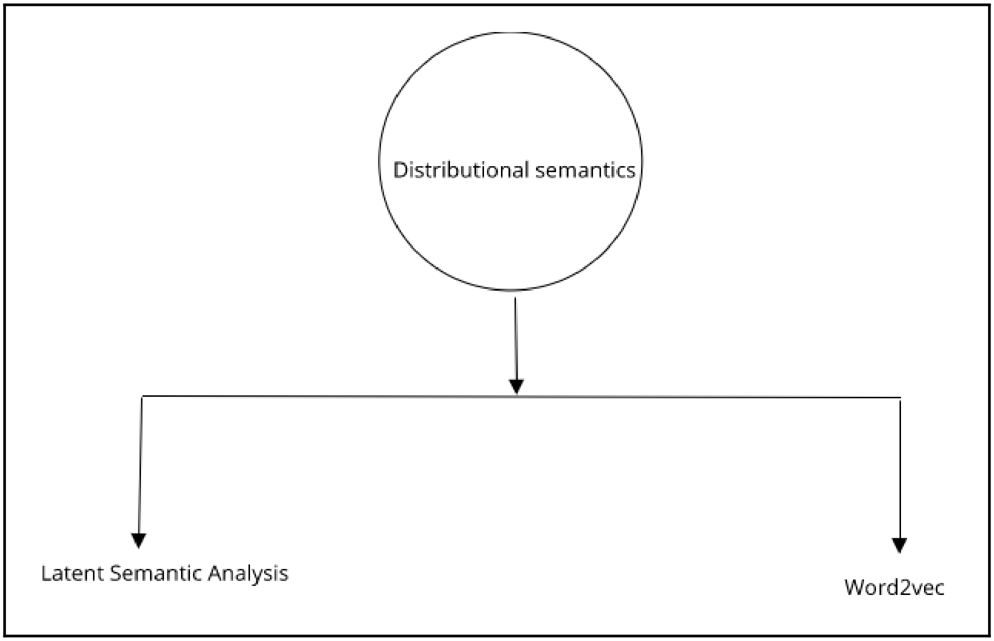
$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

Chapter 6: Advanced Feature Engineering and NLP Algorithms



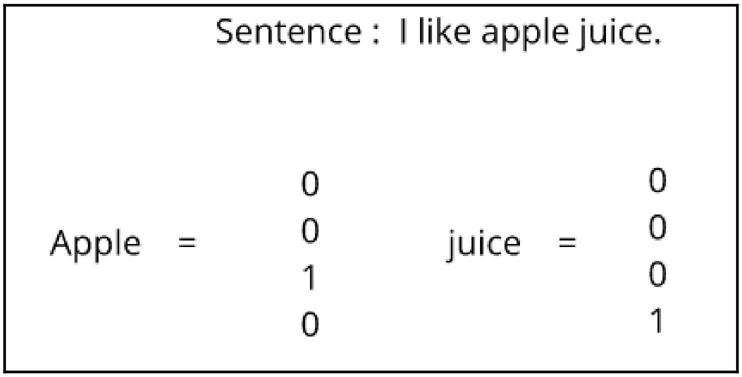




Distributional semantics

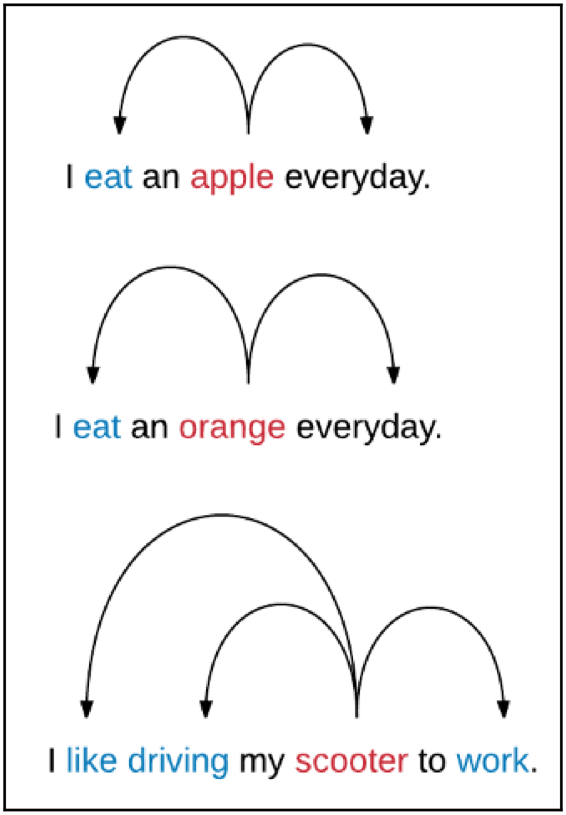
Latent Semantic Analysis

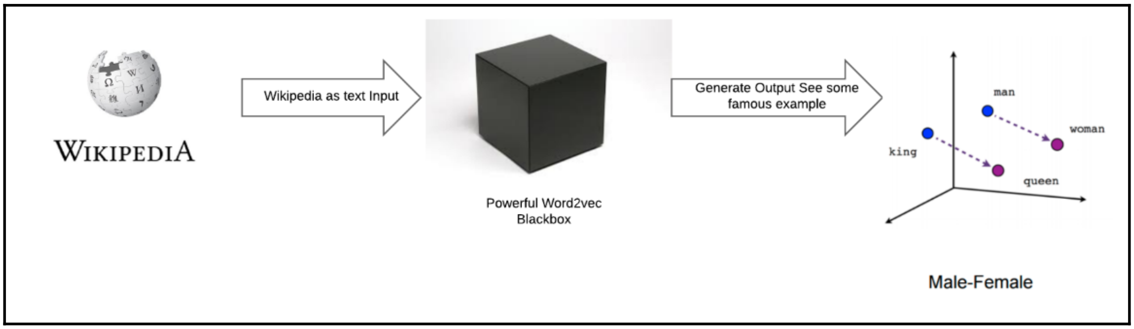
Word2vec

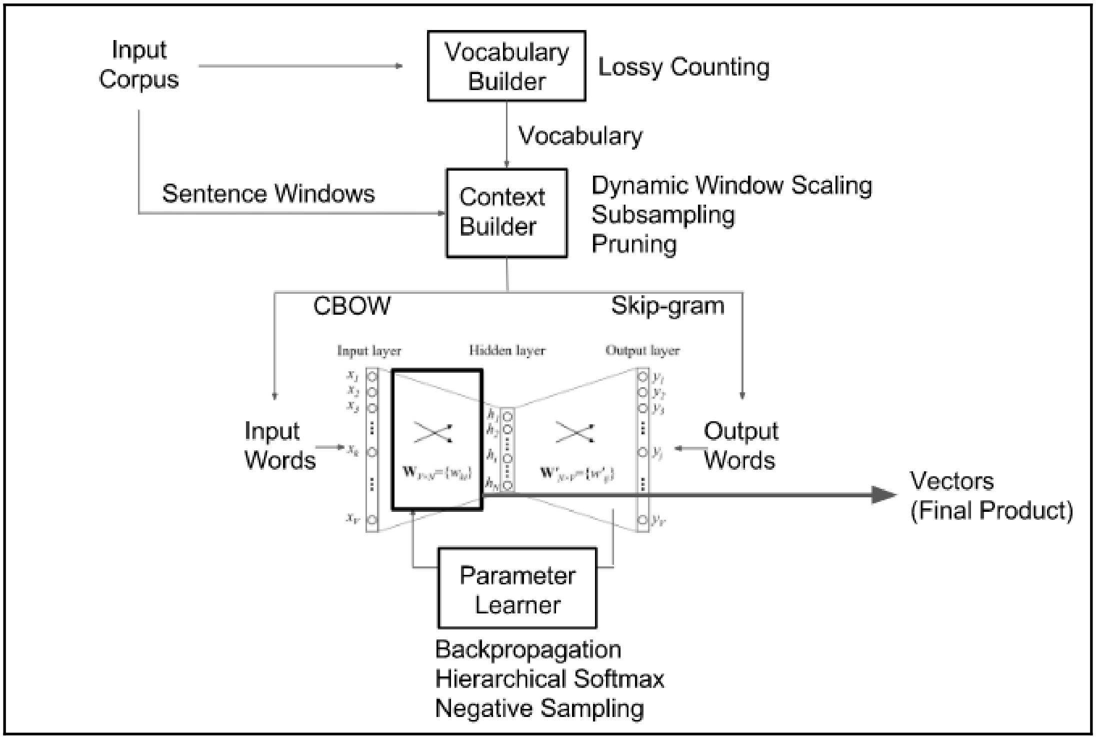


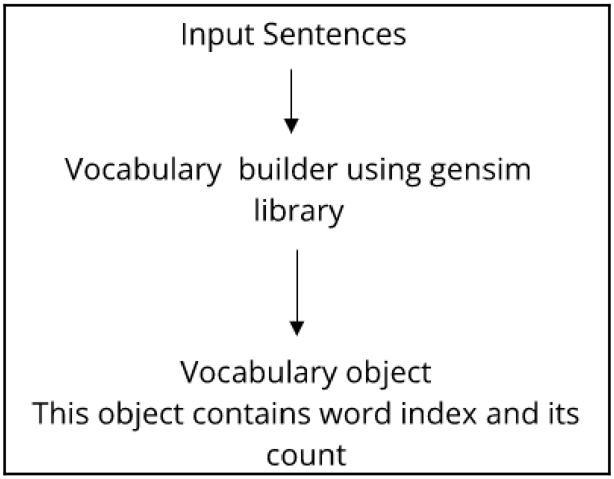
Sentence : I like apple juice.

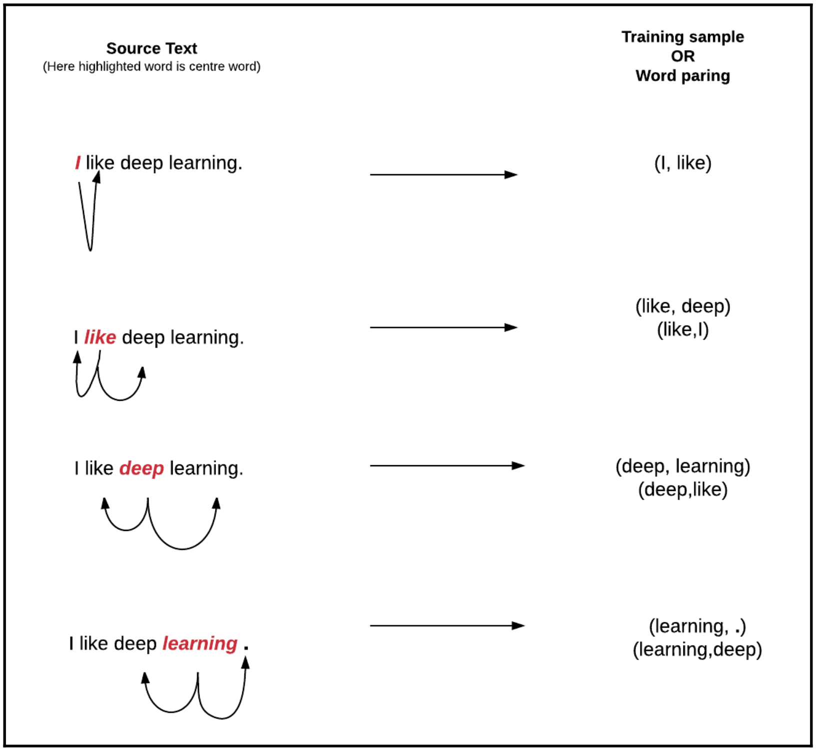
$$\text{Apple} = \begin{matrix} 0 \\ 0 \\ 1 \\ 0 \end{matrix} \qquad \text{juice} = \begin{matrix} 0 \\ 0 \\ 0 \\ 1 \end{matrix}$$

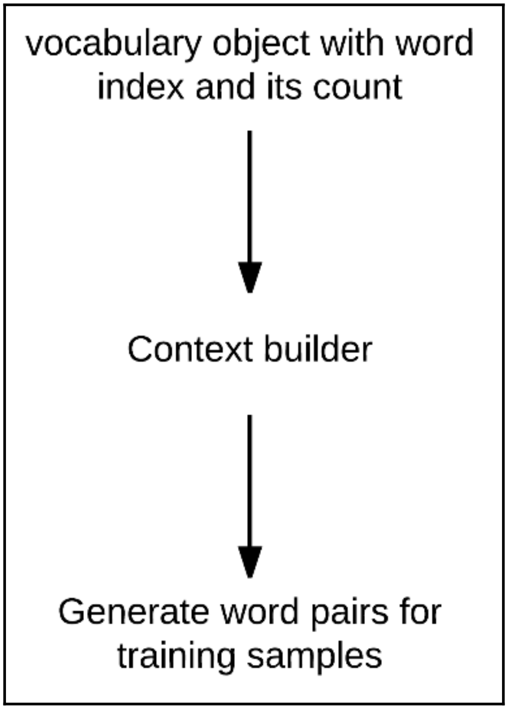

I eat an apple everyday.
I eat an orange everyday.
I like driving my scooter to work.

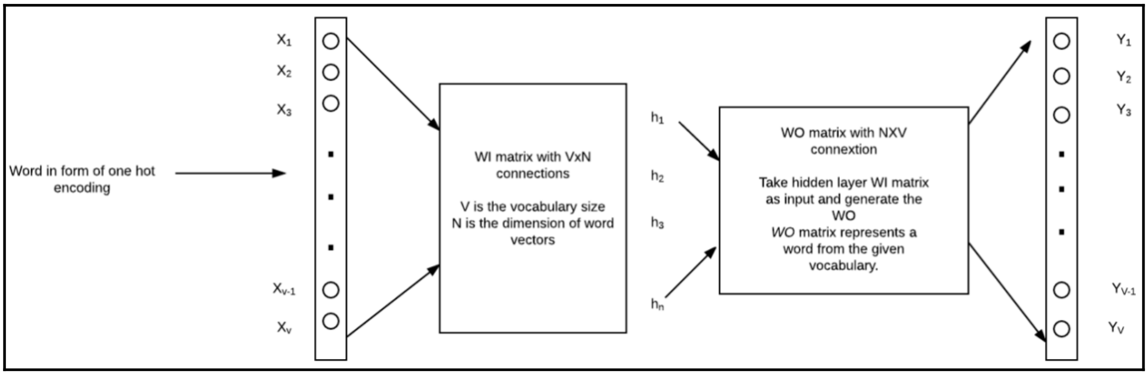

WIKIPEDIA
Wikipedia as text Input
Powerful Word2vec Blackbox
Generate Output See some famous example
man
woman
king
queen
Male-Female

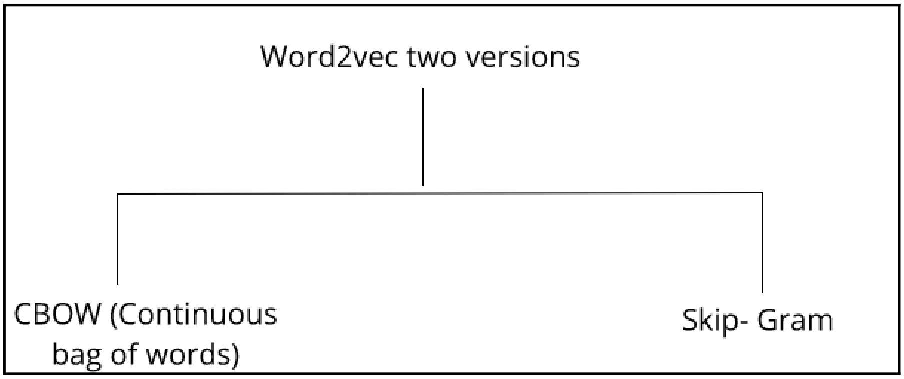

Input Corpus
Vocabulary Builder
Lossy Counting
Vocabulary
Sentence Windows
Context Builder
Dynamic Window Scaling
Subsampling
Pruning
CBOW
Skip-gram
Input layer
Hidden layer
Output layer
Input Words
Output Words
Vectors (Final Product)
Parameter Learner
Backpropagation
Hierarchical Softmax
Negative Sampling


Input Sentences
Vocabulary builder using gensim library
Vocabulary object
This object contains word index and its count

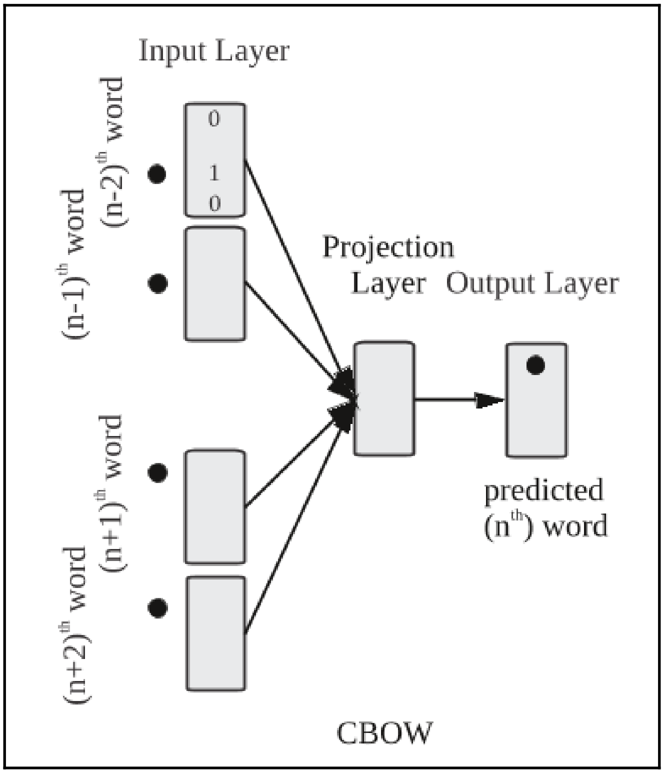

Source Text
(Here highlighted word is centre word)
Training sample
OR
Word paring
I like deep learning.
(I, like)
I like deep learning.
(like, deep)
(like,I)
I like deep learning.
(deep, learning)
(deep,like)
I like deep learning .
(learning, .)
(learning,deep)


vocabulary object with word index and its count
Context builder
Generate word pairs for training samples

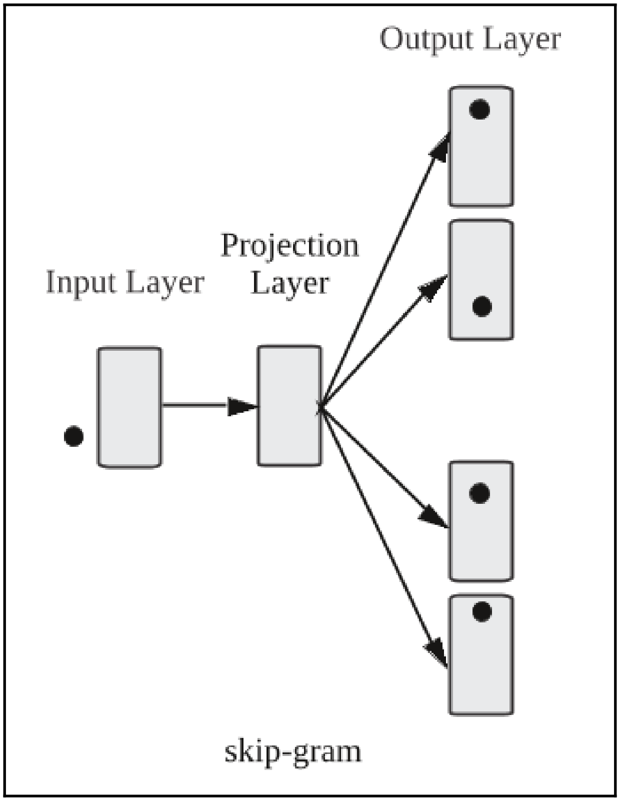

Word in form of one hot encoding
X_1
X_2
X_3
X_{v-1}
X_v
WI matrix with VxN connections
V is the vocabulary size
N is the dimension of word vectors
h_1
h_2
h_3
h_n
WO matrix with NXV connextion
Take hidden layer WI matrix as input and generate the WO
WO matrix represents a word from the given vocabulary.
Y_1
Y_2
Y_3
Y_{V-1}
Y_V

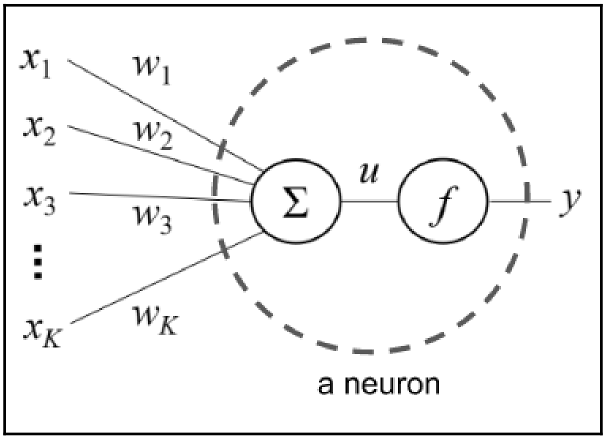


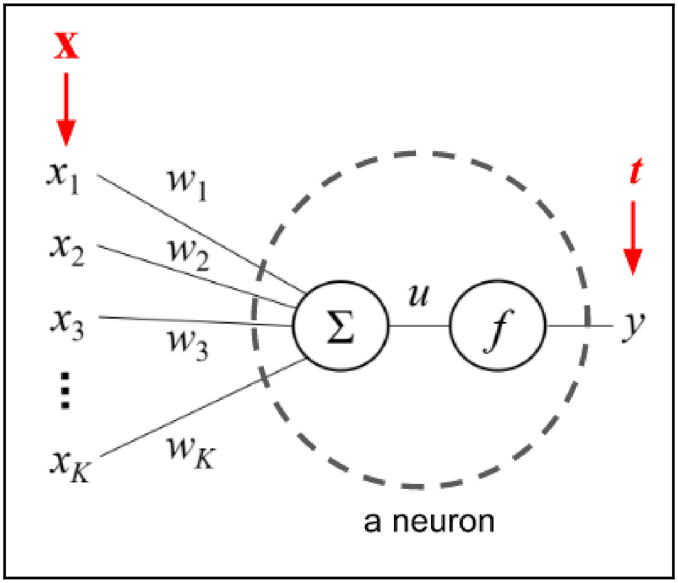
Word2vec two versions
CBOW (Continuous bag of words)
Skip- Gram

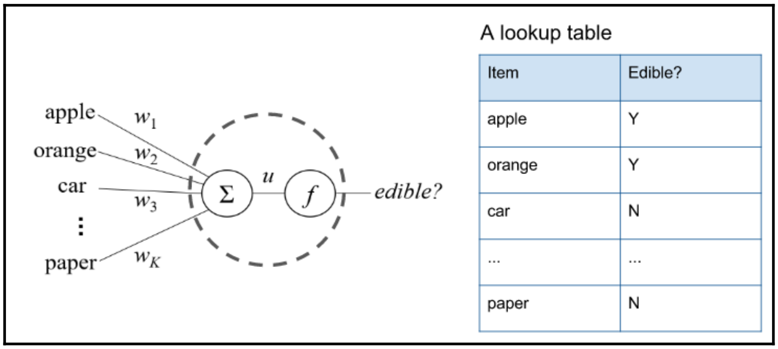


Input Layer
(n-2)th word
(n-1)th word
(n+1)th word
(n+2)th word
0
1
0
Projection Layer
Output Layer
predicted (nth) word
CBOW


Output Layer
Projection Layer
Input Layer
skip-gram

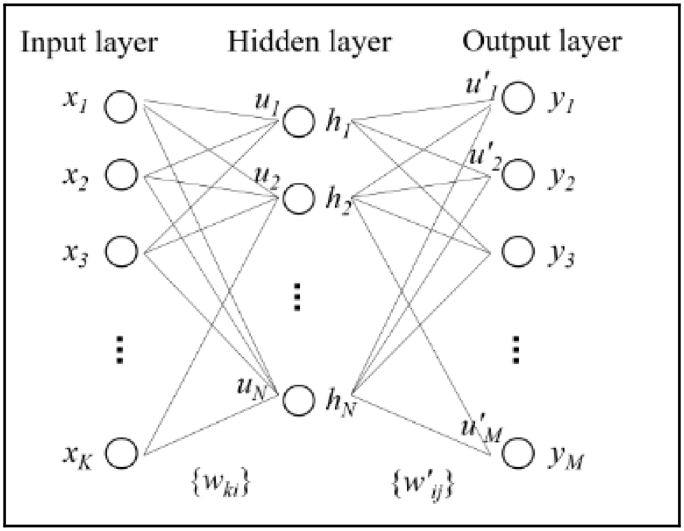

x_1
w_1
x_2
w_2
x_3
w_3
x_K
w_K
Σ
u
f
y
a neuron

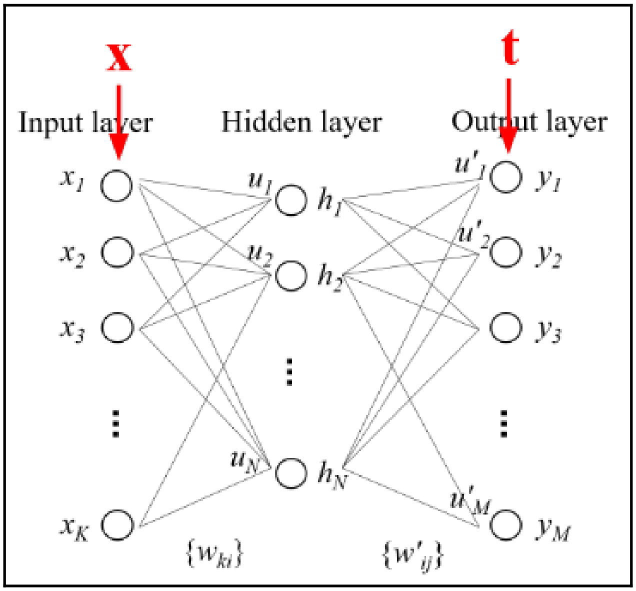


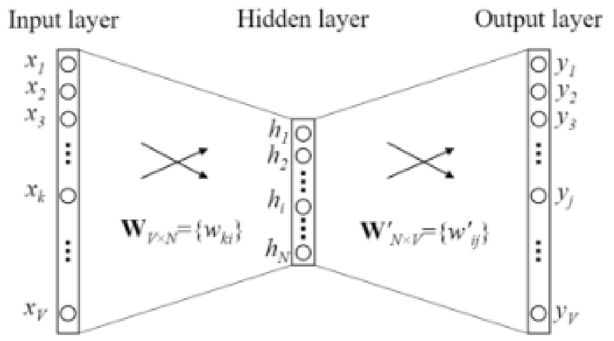


A lookup table

Item	Edible?
apple	Y
orange	Y
car	N
...	...
paper	N


Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_K
u_1
u_2
u_N
h_1
h_2
h_N
u'_1
u'_2
u'_M
y_1
y_2
y_3
y_M
$\{w_{ki}\}$
$\{w'_{ij}\}$

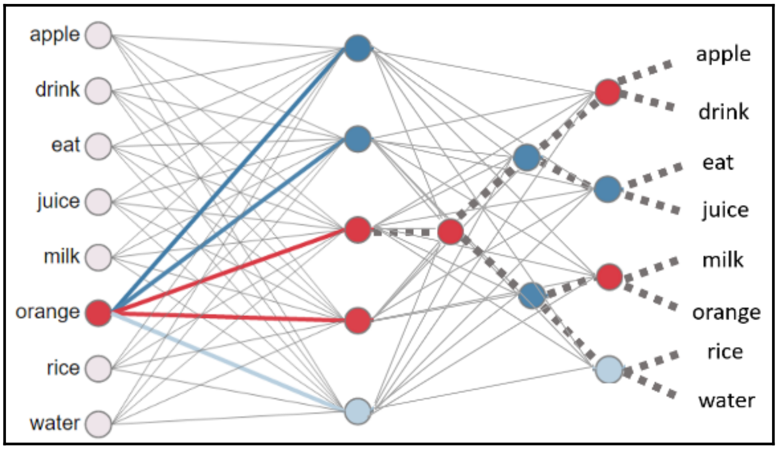

x
t
Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_K
u_1
u_2
u_N
h_1
h_2
h_N
u'_1
u'_2
u'_M
y_1
y_2
y_3
y_M
$\{w_{ki}\}$
$\{w'_{ij}\}$

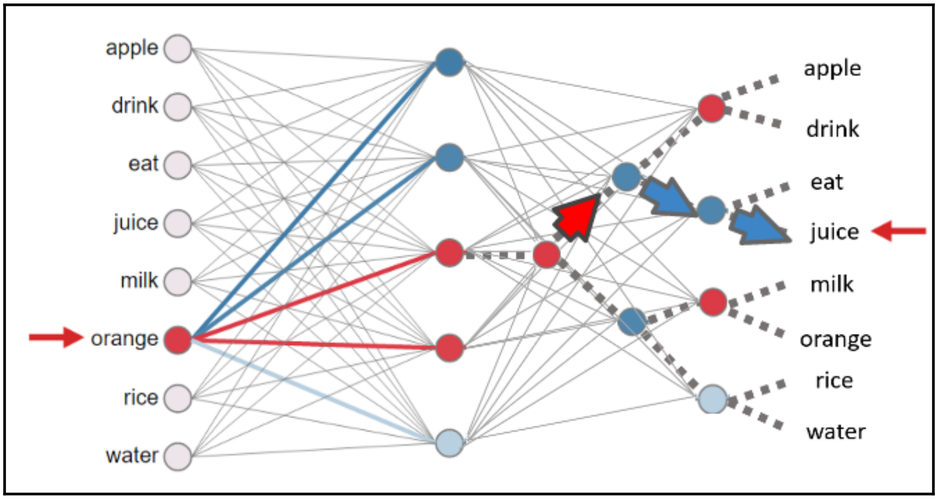


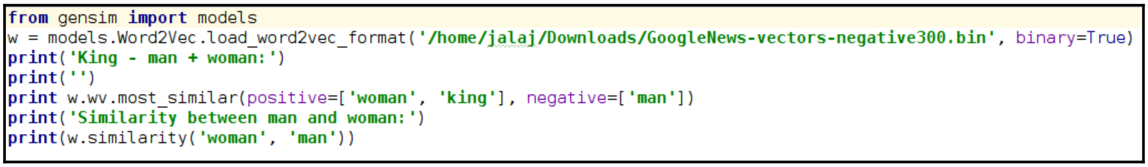
$$\mathbf{h} = \mathbf{x}^T\mathbf{W} := \mathbf{v}_{w_I}$$

$$u_j = {\mathbf{v}'_{w_j}}^T \cdot \mathbf{h}$$

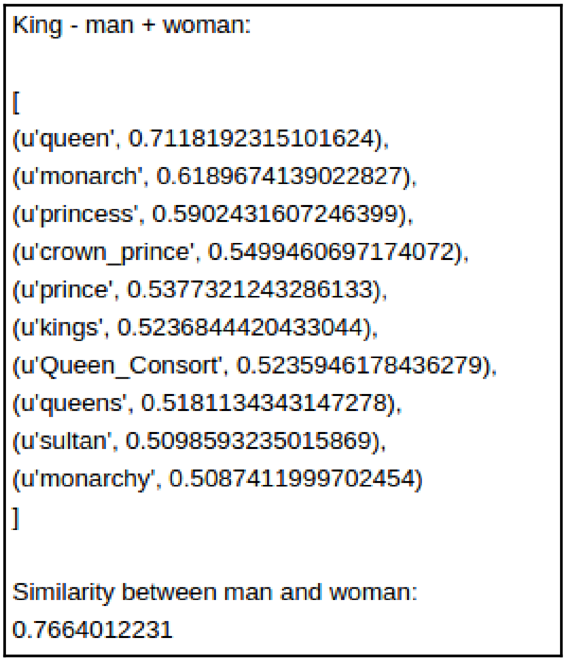
$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})}$$



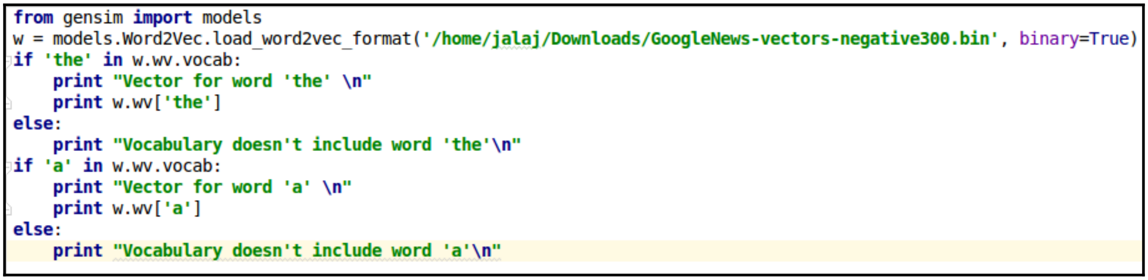




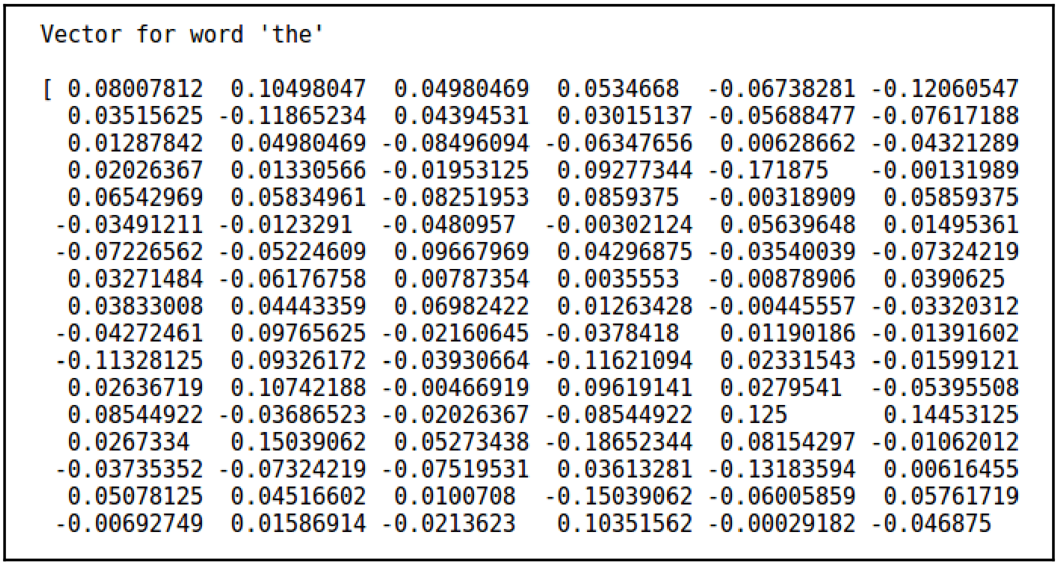
```
from gensim import models
w = models.Word2Vec.load_word2vec_format('/home/jalaj/Downloads/GoogleNews-vectors-negative300.bin', binary=True)
print('King - man + woman:')
print('')
print w.wv.most_similar(positive=['woman', 'king'], negative=['man'])
print('Similarity between man and woman:')
print(w.similarity('woman', 'man'))
```



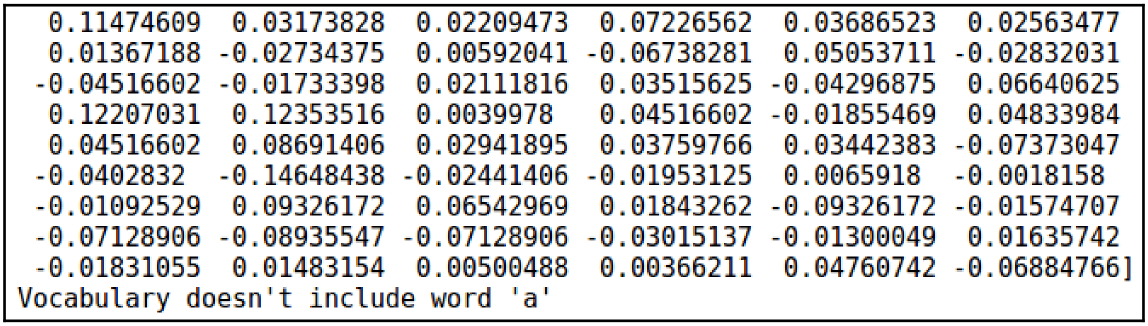
```
King - man + woman:

[
(u'queen', 0.7118192315101624),
(u'monarch', 0.6189674139022827),
(u'princess', 0.5902431607246399),
(u'crown_prince', 0.5499460697174072),
(u'prince', 0.5377321243286133),
(u'kings', 0.5236844420433044),
(u'Queen_Consort', 0.5235946178436279),
(u'queens', 0.5181134343147278),
(u'sultan', 0.5098593235015869),
(u'monarchy', 0.5087411999702454)
]

Similarity between man and woman:
0.7664012231
```



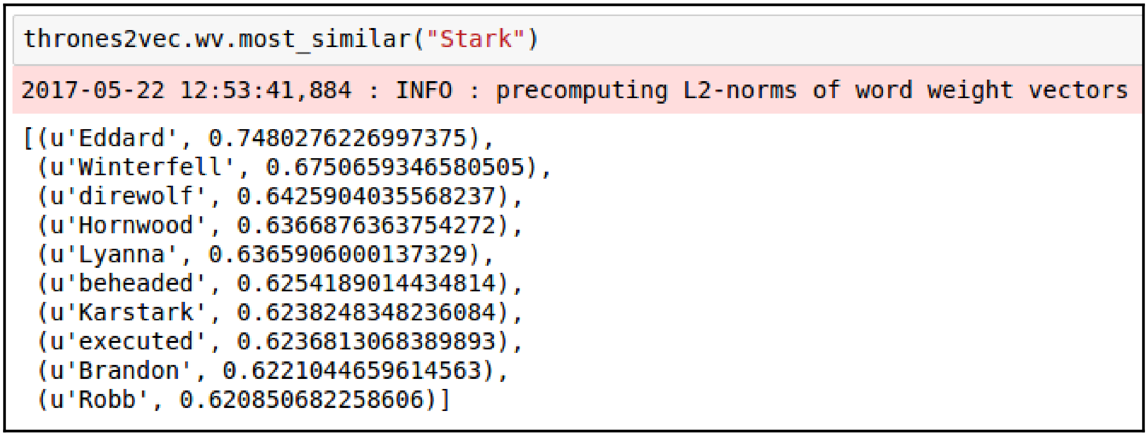
```
from gensim import models
w = models.Word2Vec.load_word2vec_format('/home/jalaj/Downloads/GoogleNews-vectors-negative300.bin', binary=True)
if 'the' in w.wv.vocab:
    print "Vector for word 'the' \n"
    print w.wv['the']
else:
    print "Vocabulary doesn't include word 'the'\n"
if 'a' in w.wv.vocab:
    print "Vector for word 'a' \n"
    print w.wv['a']
else:
    print "Vocabulary doesn't include word 'a'\n"
```



```
Vector for word 'the'

[ 0.08007812  0.10498047  0.04980469  0.0534668  -0.06738281 -0.12060547
  0.03515625 -0.11865234  0.04394531  0.03015137 -0.05688477 -0.07617188
  0.01287842  0.04980469 -0.08496094 -0.06347656  0.00628662 -0.04321289
  0.02026367  0.01330566 -0.01953125  0.09277344 -0.171875   -0.00131989
  0.06542969  0.05834961 -0.08251953  0.0859375  -0.00318909  0.05859375
 -0.03491211 -0.0123291  -0.0480957  -0.00302124  0.05639648  0.01495361
 -0.07226562 -0.05224609  0.09667969  0.04296875 -0.03540039 -0.07324219
  0.03271484 -0.06176758  0.00787354  0.0035553  -0.00878906  0.0390625
  0.03833008  0.04443359  0.06982422  0.01263428 -0.00445557 -0.03320312
 -0.04272461  0.09765625 -0.02160645 -0.0378418   0.01190186 -0.01391602
 -0.11328125  0.09326172 -0.03930664 -0.11621094  0.02331543 -0.01599121
  0.02636719  0.10742188 -0.00466919  0.09619141  0.0279541  -0.05395508
  0.08544922 -0.03686523 -0.02026367 -0.08544922  0.125       0.14453125
  0.0267334   0.15039062  0.05273438 -0.18652344  0.08154297 -0.01062012
 -0.03735352 -0.07324219 -0.07519531  0.03613281 -0.13183594  0.00616455
  0.05078125  0.04516602  0.0100708  -0.15039062 -0.06005859  0.05761719
 -0.00692749  0.01586914 -0.0213623   0.10351562 -0.00029182 -0.046875
```



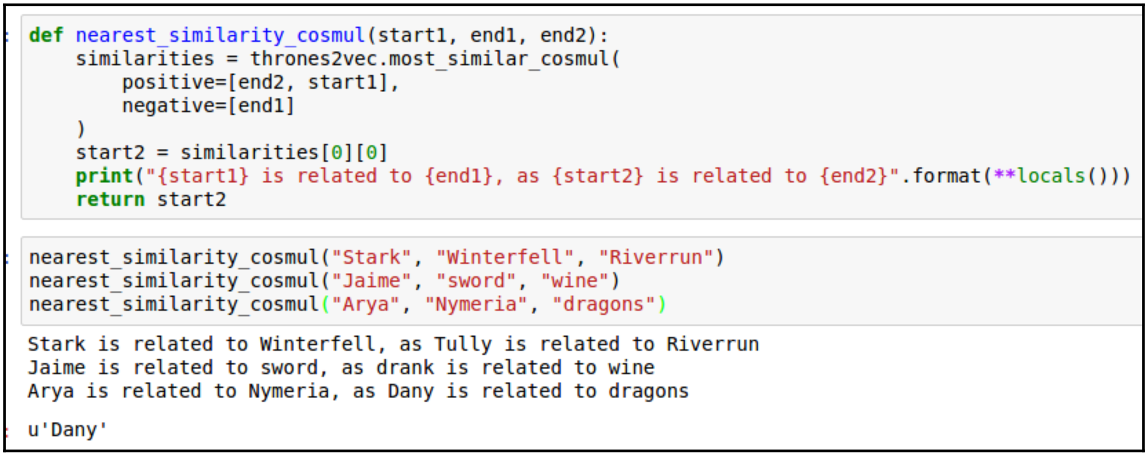
```
  0.11474609  0.03173828  0.02209473  0.07226562  0.03686523  0.02563477
  0.01367188 -0.02734375  0.00592041 -0.06738281  0.05053711 -0.02832031
 -0.04516602 -0.01733398  0.02111816  0.03515625 -0.04296875  0.06640625
  0.12207031  0.12353516  0.0039978   0.04516602 -0.01855469  0.04833984
  0.04516602  0.08691406  0.02941895  0.03759766  0.03442383 -0.07373047
 -0.0402832  -0.14648438 -0.02441406 -0.01953125  0.0065918  -0.0018158
 -0.01092529  0.09326172  0.06542969  0.01843262 -0.09326172 -0.01574707
 -0.07128906 -0.08935547 -0.07128906 -0.03015137 -0.01300049  0.01635742
 -0.01831055  0.01483154  0.00500488  0.00366211  0.04760742 -0.06884766]
Vocabulary doesn't include word 'a'
```



```
thrones2vec.wv.most_similar("Stark")
```

```
2017-05-22 12:53:41,884 : INFO : precomputing L2-norms of word weight vectors
```

```
[(u'Eddard', 0.7480276226997375),
 (u'Winterfell', 0.6750659346580505),
 (u'direwolf', 0.6425904035568237),
 (u'Hornwood', 0.6366876363754272),
 (u'Lyanna', 0.6365906000137329),
 (u'beheaded', 0.6254189014434814),
 (u'Karstark', 0.6238248348236084),
 (u'executed', 0.6236813068389893),
 (u'Brandon', 0.6221044659614563),
 (u'Robb', 0.620850682258606)]
```

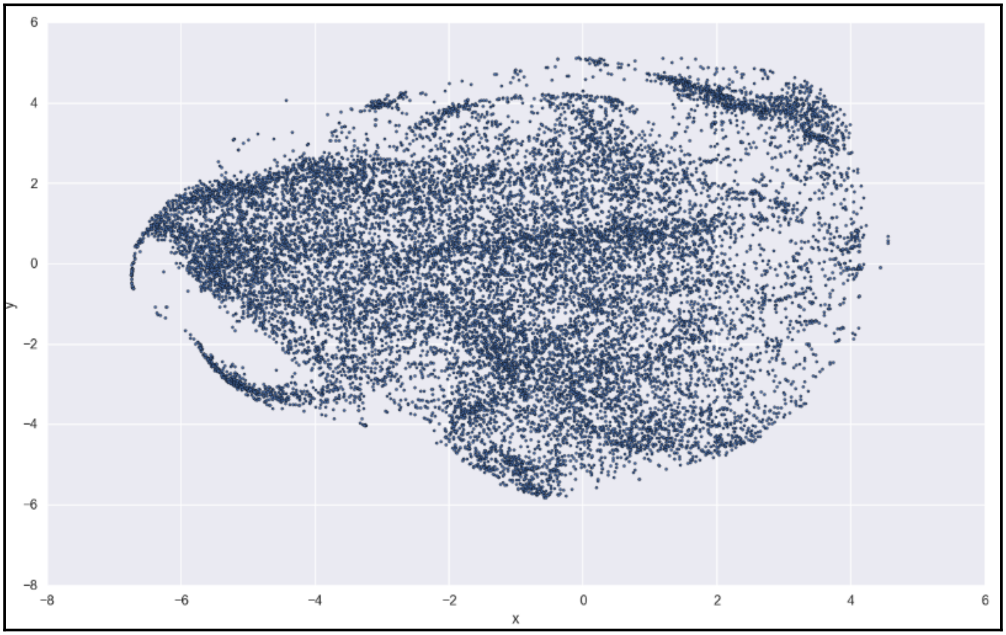


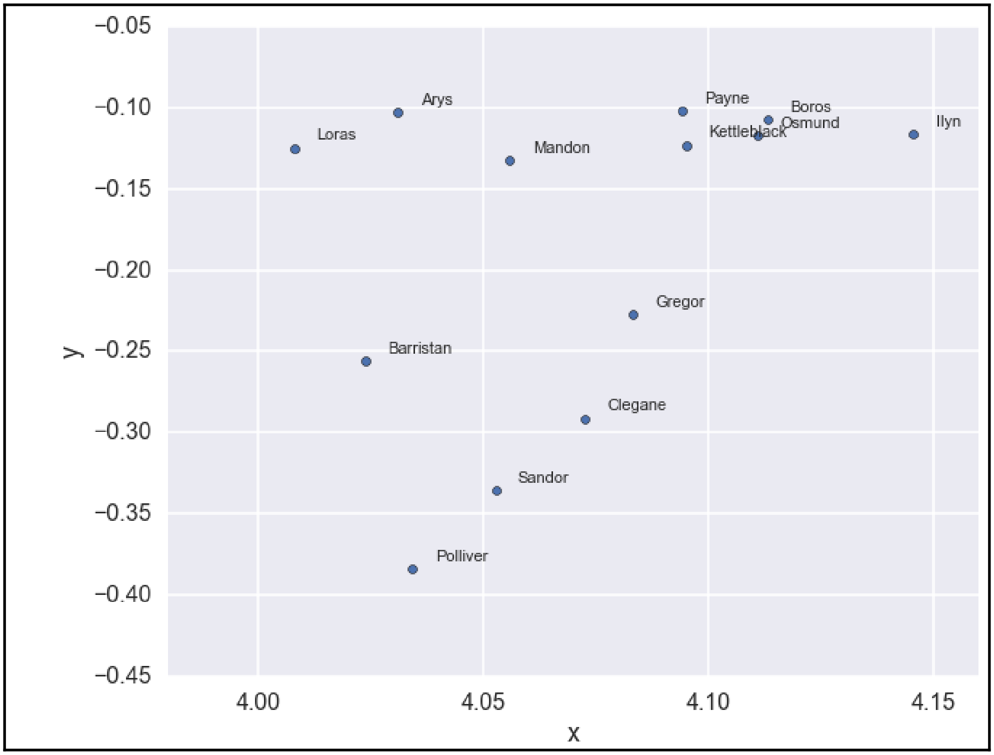
```
def nearest_similarity_cosmul(start1, end1, end2):
    similarities = thrones2vec.most_similar_cosmul(
        positive=[end2, start1],
        negative=[end1]
    )
    start2 = similarities[0][0]
    print("{start1} is related to {end1}, as {start2} is related to {end2}".format(**locals()))
    return start2
```

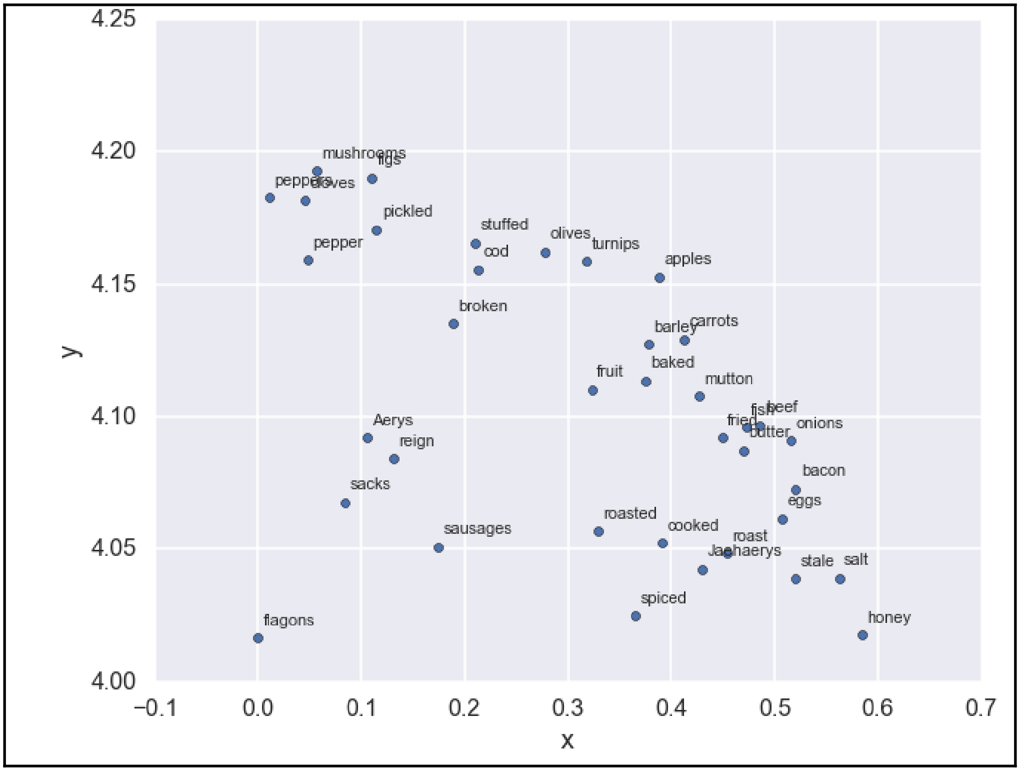
```
nearest_similarity_cosmul("Stark", "Winterfell", "Riverrun")
nearest_similarity_cosmul("Jaime", "sword", "wine")
nearest_similarity_cosmul("Arya", "Nymeria", "dragons")
```

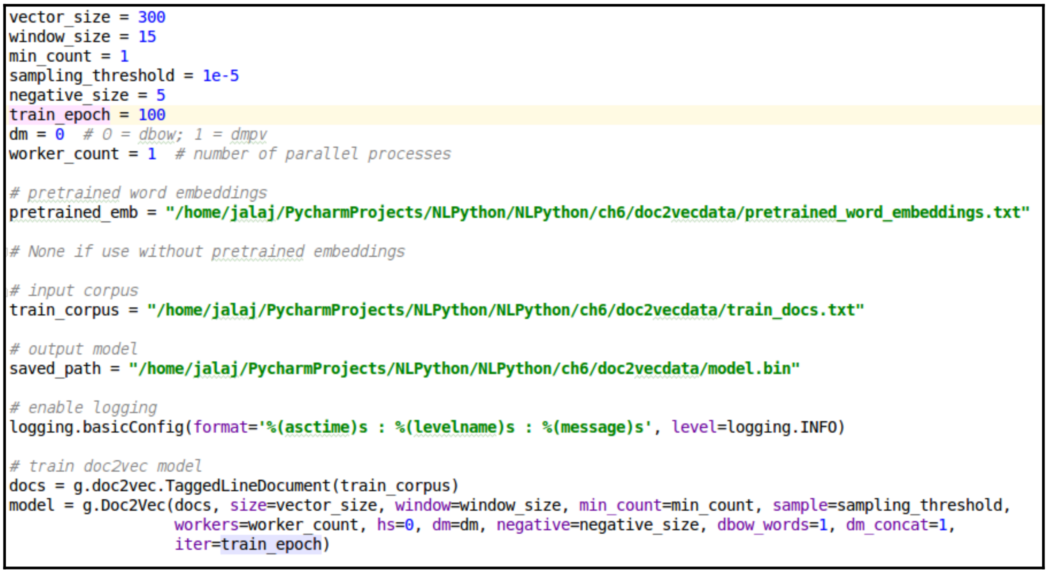
```
Stark is related to Winterfell, as Tully is related to Riverrun
Jaime is related to sword, as drank is related to wine
Arya is related to Nymeria, as Dany is related to dragons
```

```
u'Dany'
```

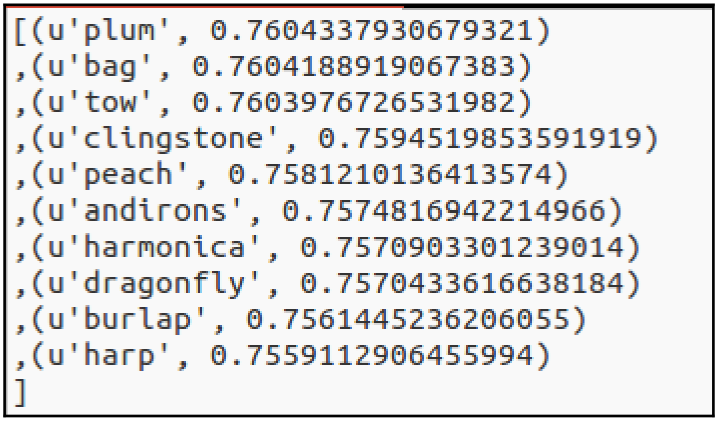

6
4
2
0
−2
−4
−6
−8
y
−8
−6
−4
−2
0
2
4
6
x


Arys
Loras
Mandon
Payne
Boros
Kettleblack
Osmund
Ilyn
Gregor
Barristan
Clegane
Sandor
Polliver
y
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35
−0.40
−0.45
4.00
4.05
4.10
4.15
x

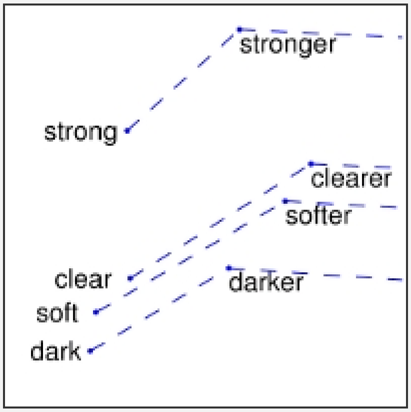




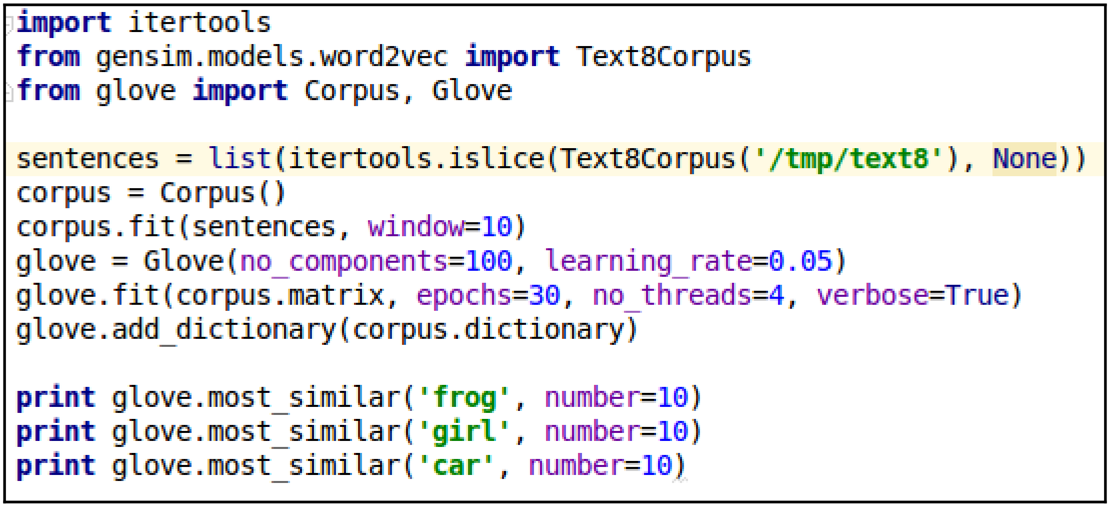
```
vector_size = 300
window_size = 15
min_count = 1
sampling_threshold = 1e-5
negative_size = 5
train_epoch = 100
dm = 0  # 0 = dbow; 1 = dmpv
worker_count = 1  # number of parallel processes

# pretrained word embeddings
pretrained_emb = "/home/jalaj/PycharmProjects/NLPython/NLPython/ch6/doc2vecdata/pretrained_word_embeddings.txt"

# None if use without pretrained embeddings

# input corpus
train_corpus = "/home/jalaj/PycharmProjects/NLPython/NLPython/ch6/doc2vecdata/train_docs.txt"

# output model
saved_path = "/home/jalaj/PycharmProjects/NLPython/NLPython/ch6/doc2vecdata/model.bin"

# enable logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

# train doc2vec model
docs = g.doc2vec.TaggedLineDocument(train_corpus)
model = g.Doc2Vec(docs, size=vector_size, window=window_size, min_count=min_count, sample=sampling_threshold,
                  workers=worker_count, hs=0, dm=dm, negative=negative_size, dbow_words=1, dm_concat=1,
                  iter=train_epoch)
```



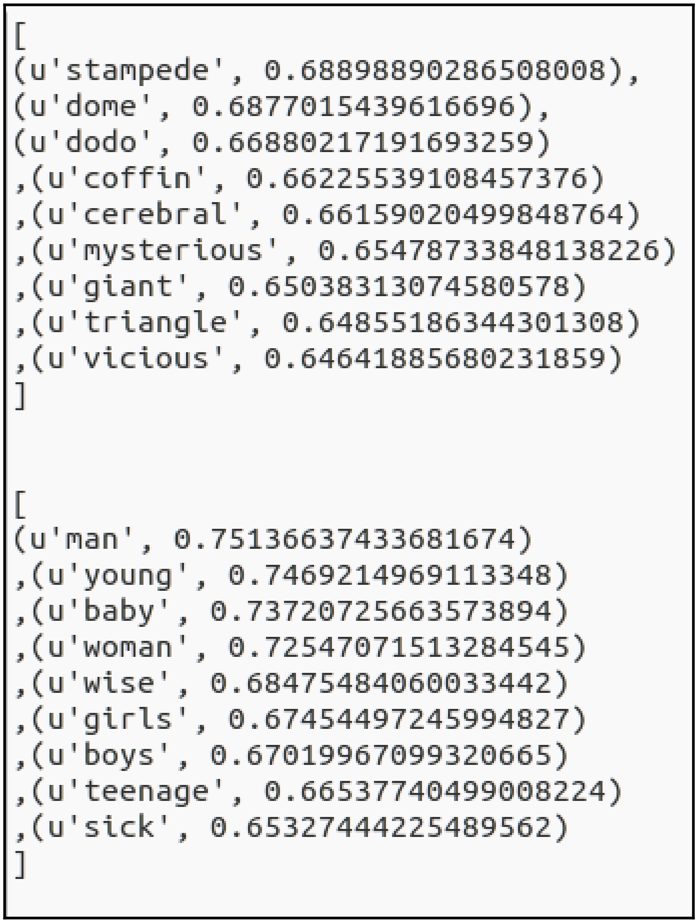
```
[(u'plum', 0.7604337930679321)
,(u'bag', 0.7604188919067383)
,(u'tow', 0.760397672653182)
,(u'clingstone', 0.7594519853591919)
,(u'peach', 0.7581210136413574)
,(u'andirons', 0.7574816942214966)
,(u'harmonica', 0.7570903301239014)
,(u'dragonfly', 0.7570433616638184)
,(u'burlap', 0.7561445236206055)
,(u'harp', 0.7559112906455994)
]
```



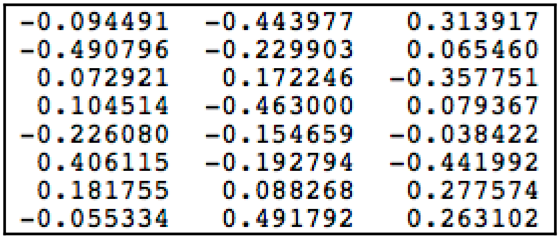
comparative - superlative



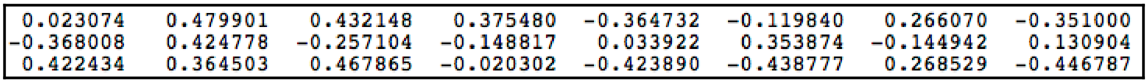
```
import itertools
from gensim.models.word2vec import Text8Corpus
from glove import Corpus, Glove

sentences = list(itertools.islice(Text8Corpus('/tmp/text8'), None))
corpus = Corpus()
corpus.fit(sentences, window=10)
glove = Glove(no_components=100, learning_rate=0.05)
glove.fit(corpus.matrix, epochs=30, no_threads=4, verbose=True)
glove.add_dictionary(corpus.dictionary)

print glove.most_similar('frog', number=10)
print glove.most_similar('girl', number=10)
print glove.most_similar('car', number=10)
```



```
[
(u'stampede', 0.6889889028650808),
(u'dome', 0.687701543961669б),
(u'dodo', 0.6688021719169325 9)
,(u'coffin', 0.66225539108457376)
,(u'cerebral', 0.6615902049984876 4)
,(u'mysterious', 0.65478733848138226)
,(u'giant', 0.6503831307458057 8)
,(u'triangle', 0.64855186344301308)
,(u'vicious', 0.64641885680231859)
]


[
(u'man', 0.75136637433681674)
,(u'young', 0.7469214969113348)
,(u'baby', 0.73720725663573894)
,(u'woman', 0.72547071513284545)
,(u'wise', 0.68475484060033442)
,(u'girls', 0.67454497245994827)
,(u'boys', 0.67019967099320665)
,(u'teenage', 0.66537740499008224)
,(u'sick', 0.65327444225489562)
]
```

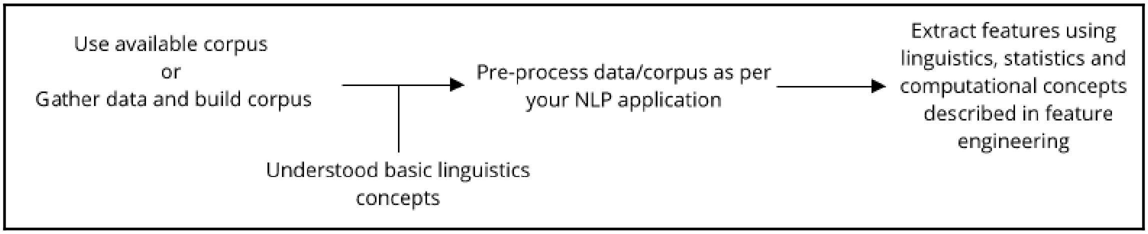


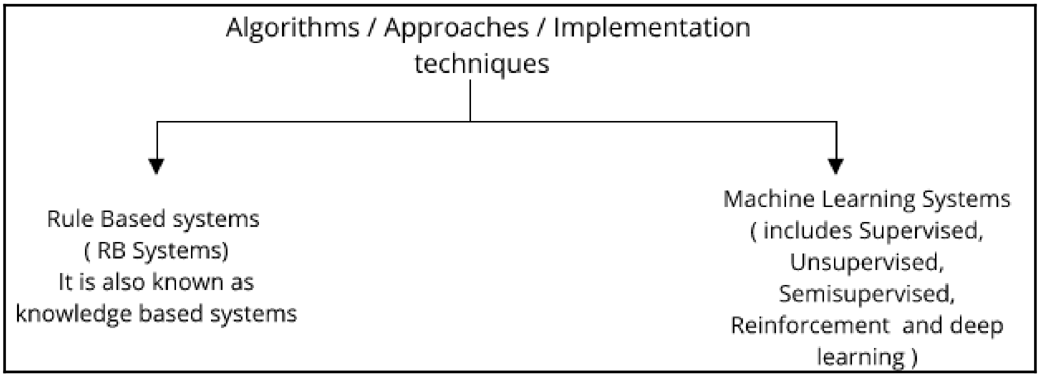
```
-0.094491  -0.443977   0.313917
-0.490796  -0.229903   0.065460
 0.072921   0.172246  -0.357751
 0.104514  -0.463000   0.079367
-0.226080  -0.154659  -0.038422
 0.406115  -0.192794  -0.441992
 0.181755   0.088268   0.277574
-0.055334   0.491792   0.263102
```

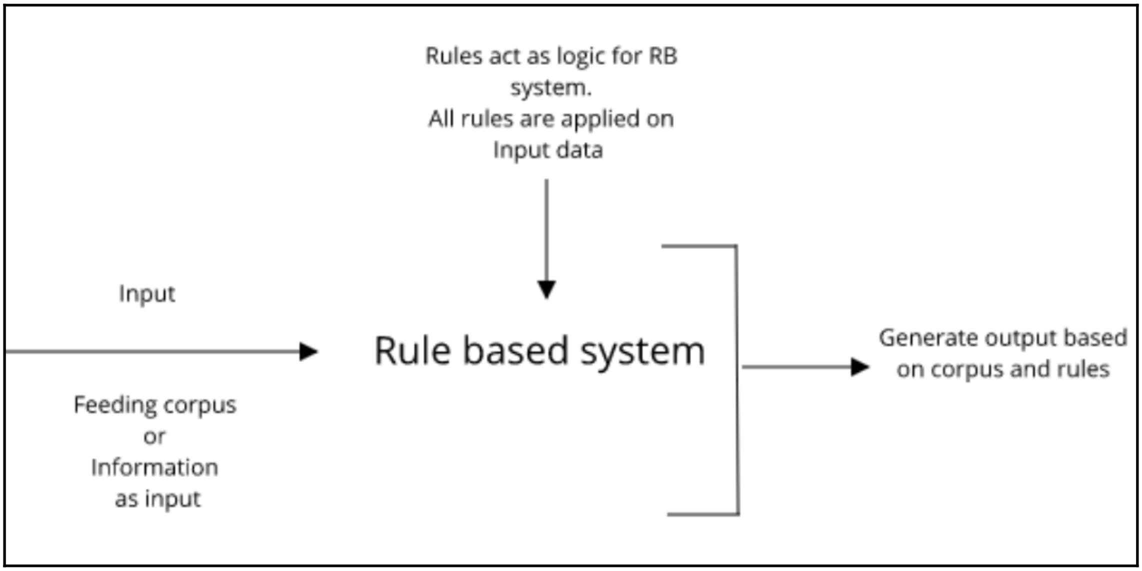


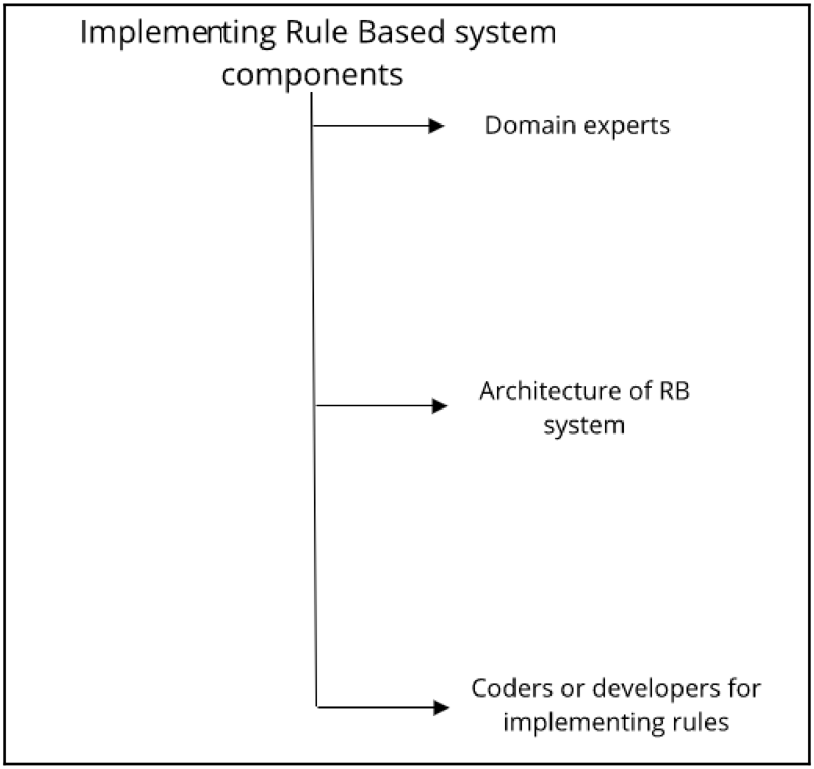
```
 0.023074   0.479901   0.432148   0.375480  -0.364732  -0.119840   0.266070  -0.351000
-0.368008   0.424778  -0.257104  -0.148817   0.033922   0.353874  -0.144942   0.130904
 0.422434   0.364503   0.467865  -0.020302  -0.423890  -0.438777   0.268529  -0.446787
```

Chapter 7: Rule-Based System for NLP


Use available corpus
or
Gather data and build corpus
Understood basic linguistics concepts
Pre-process data/corpus as per your NLP application
Extract features using linguistics, statistics and computational concepts described in feature engineering


Algorithms / Approaches / Implementation techniques
Rule Based systems (RB Systems) It is also known as knowledge based systems
Machine Learning Systems (includes Supervised, Unsupervised, Semisupervised, Reinforcement and deep learning)


Rules act as logic for RB system.
All rules are applied on Input data
Input
Feeding corpus
or
Information
as input
Rule based system
Generate output based on corpus and rules

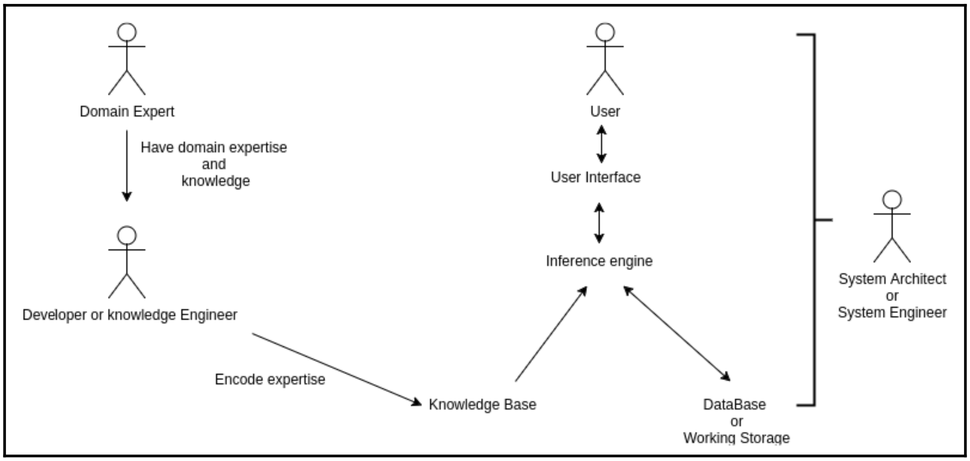


Implementing Rule Based system components

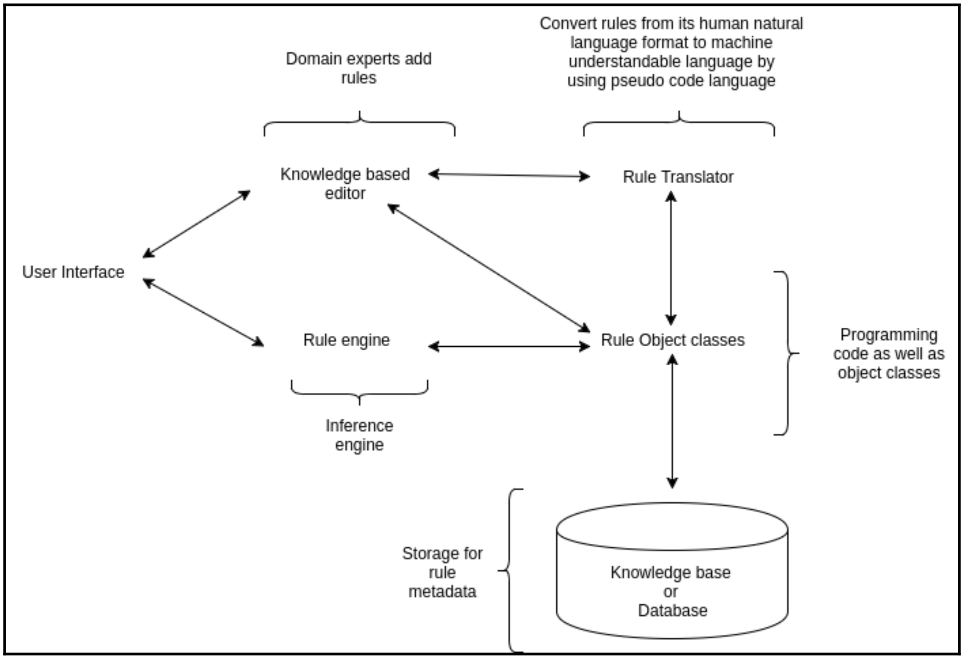
Domain experts

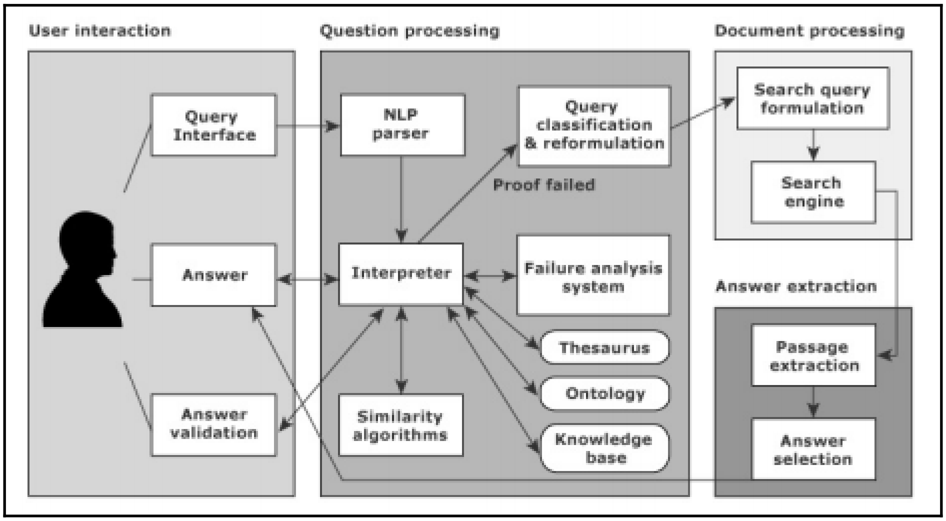
Architecture of RB system

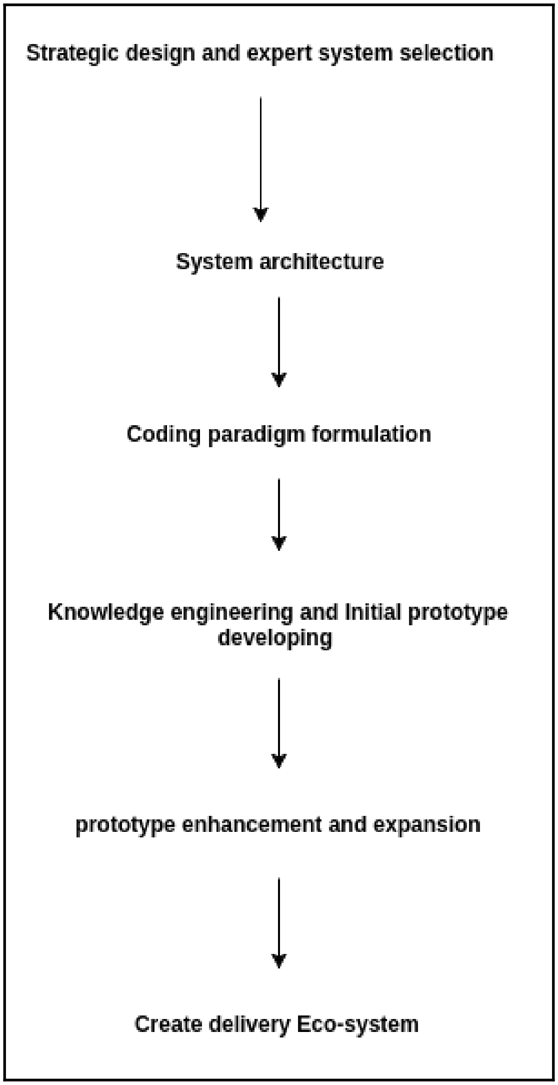
Coders or developers for implementing rules

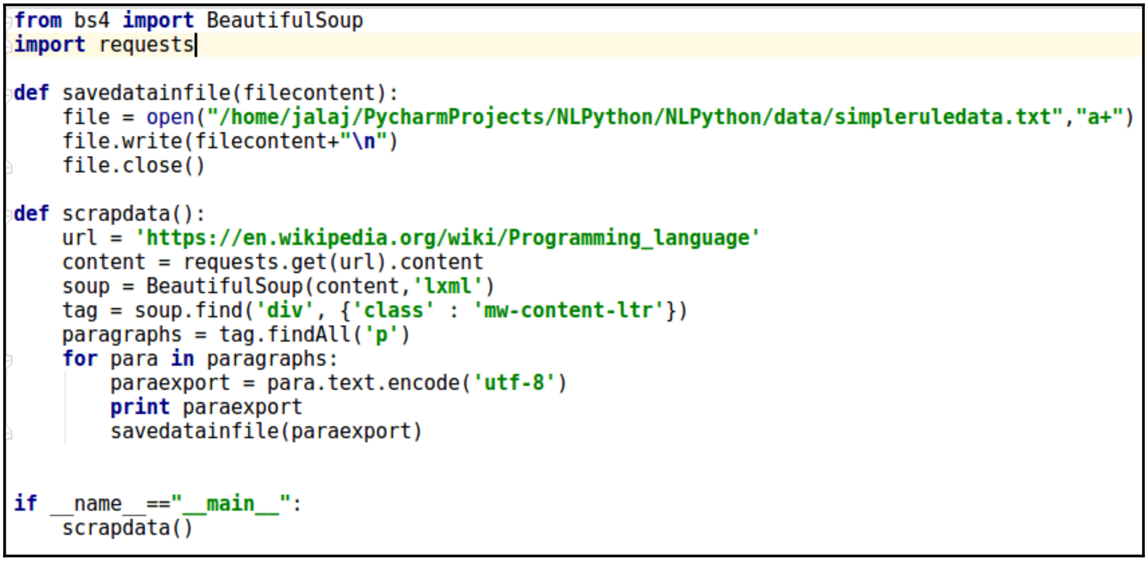


Domain Expert
Have domain expertise and knowledge
Developer or knowledge Engineer
Encode expertise
Knowledge Base
User
User Interface
Inference engine
DataBase or Working Storage
System Architect or System Engineer

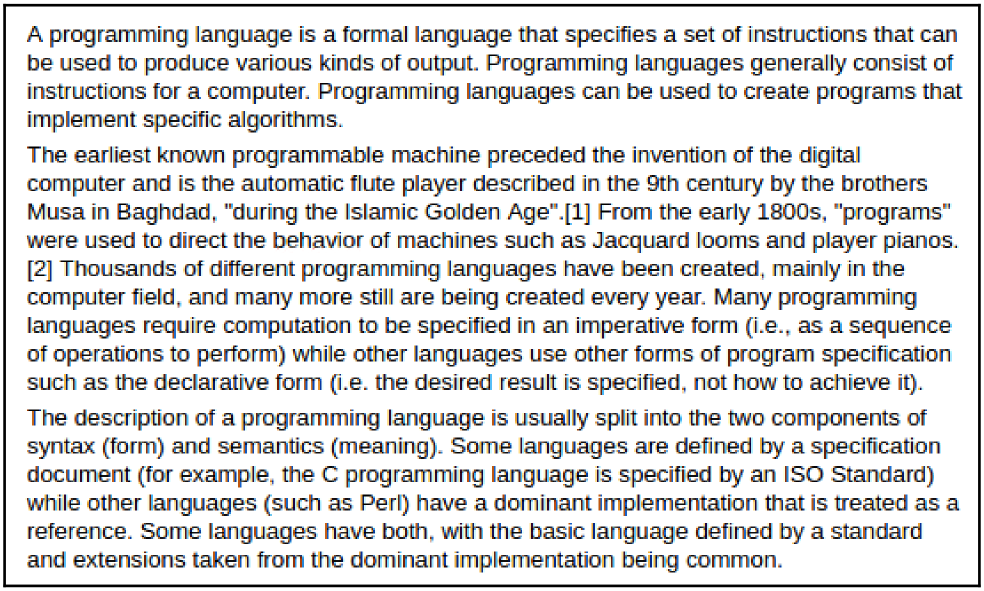

Domain experts add rules
Convert rules from its human natural language format to machine understandable language by using pseudo code language
Knowledge based editor
Rule Translator
User Interface
Rule engine
Rule Object classes
Programming code as well as object classes
Inference engine
Storage for rule metadata
Knowledge base or Database


User interaction
Question processing
Document processing
Query Interface
NLP parser
Query classification & reformulation
Search query formulation
Search engine
Proof failed
Answer
Interpreter
Failure analysis system
Answer extraction
Thesaurus
Ontology
Knowledge base
Passage extraction
Answer validation
Similarity algorithms
Answer selection


Strategic design and expert system selection
System architecture
Coding paradigm formulation
Knowledge engineering and Initial prototype developing
prototype enhancement and expansion
Create delivery Eco-system



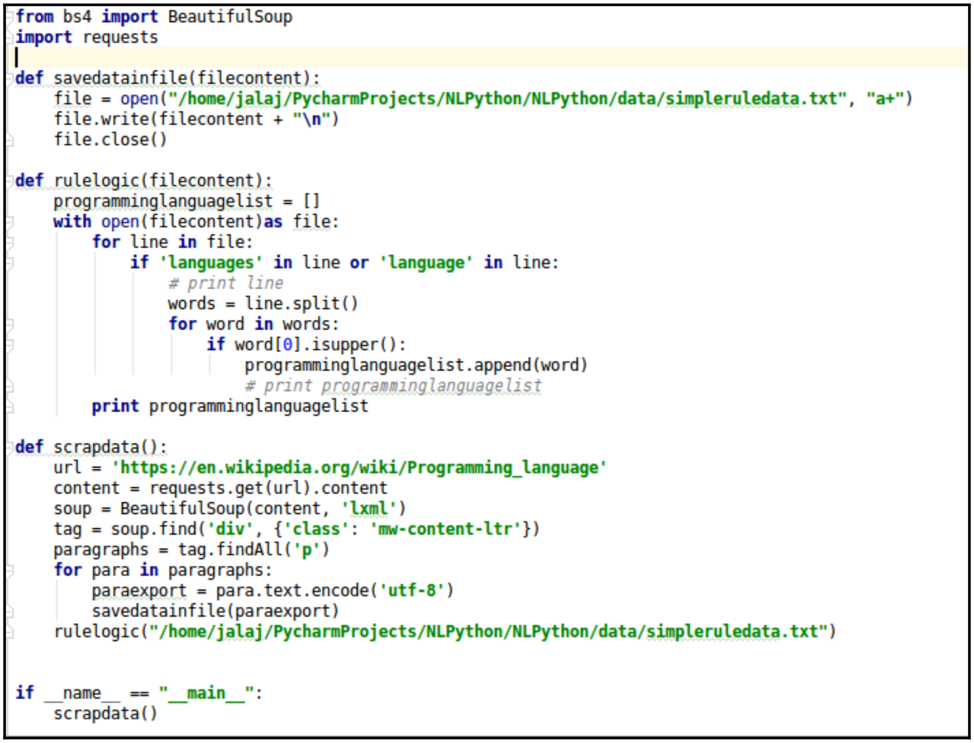
```
from bs4 import BeautifulSoup
import requests

def savedatainfile(filecontent):
    file = open("/home/jalaj/PycharmProjects/NLPython/NLPython/data/simpleruledata.txt","a+")
    file.write(filecontent+"\n")
    file.close()

def scrapdata():
    url = 'https://en.wikipedia.org/wiki/Programming_language'
    content = requests.get(url).content
    soup = BeautifulSoup(content,'lxml')
    tag = soup.find('div', {'class' : 'mw-content-ltr'})
    paragraphs = tag.findAll('p')
    for para in paragraphs:
        paraexport = para.text.encode('utf-8')
        print paraexport
        savedatainfile(paraexport)


if __name__=="__main__":
    scrapdata()
```



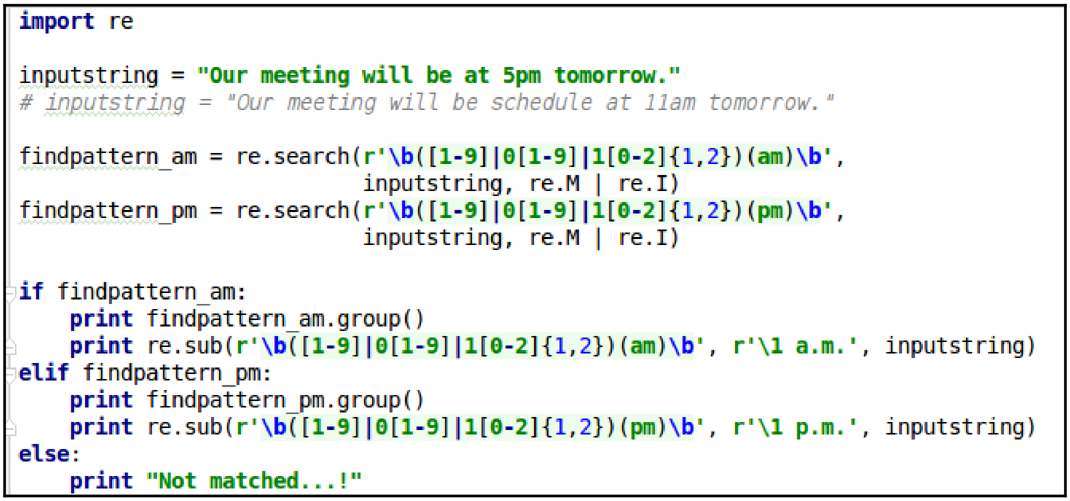
A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output. Programming languages generally consist of instructions for a computer. Programming languages can be used to create programs that implement specific algorithms.

The earliest known programmable machine preceded the invention of the digital computer and is the automatic flute player described in the 9th century by the brothers Musa in Baghdad, "during the Islamic Golden Age".[1] From the early 1800s, "programs" were used to direct the behavior of machines such as Jacquard looms and player pianos.[2] Thousands of different programming languages have been created, mainly in the computer field, and many more still are being created every year. Many programming languages require computation to be specified in an imperative form (i.e., as a sequence of operations to perform) while other languages use other forms of program specification such as the declarative form (i.e. the desired result is specified, not how to achieve it).

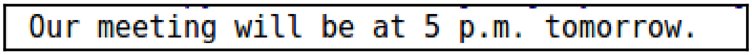
The description of a programming language is usually split into the two components of syntax (form) and semantics (meaning). Some languages are defined by a specification document (for example, the C programming language is specified by an ISO Standard) while other languages (such as Perl) have a dominant implementation that is treated as a reference. Some languages have both, with the basic language defined by a standard and extensions taken from the dominant implementation being common.



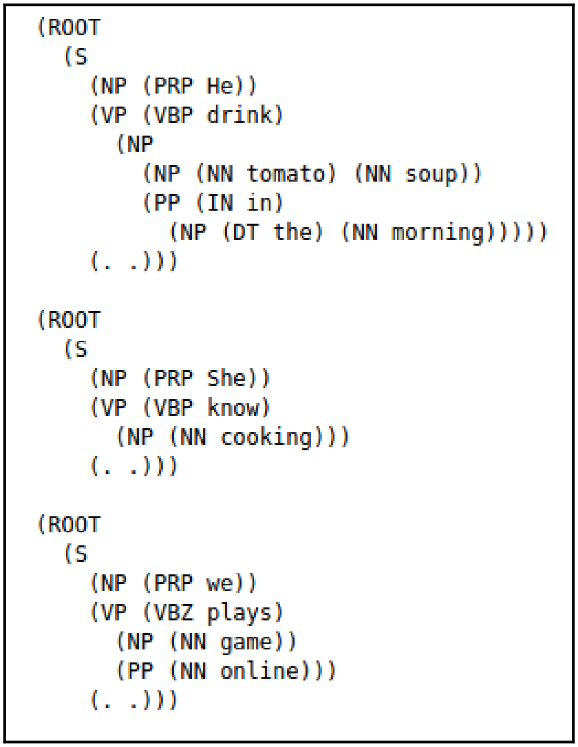
```
from bs4 import BeautifulSoup
import requests

def savedatainfile(filecontent):
    file = open("/home/jalaj/PycharmProjects/NLPython/NLPython/data/simpleruledata.txt", "a+")
    file.write(filecontent + "\n")
    file.close()

def rulelogic(filecontent):
    programminglanguagelist = []
    with open(filecontent)as file:
        for line in file:
            if 'languages' in line or 'language' in line:
                # print line
                words = line.split()
                for word in words:
                    if word[0].isupper():
                        programminglanguagelist.append(word)
                        # print programminglanguagelist
        print programminglanguagelist

def scrapdata():
    url = 'https://en.wikipedia.org/wiki/Programming_language'
    content = requests.get(url).content
    soup = BeautifulSoup(content, 'lxml')
    tag = soup.find('div', {'class': 'mw-content-ltr'})
    paragraphs = tag.findAll('p')
    for para in paragraphs:
        paraexport = para.text.encode('utf-8')
        savedatainfile(paraexport)
    rulelogic("/home/jalaj/PycharmProjects/NLPython/NLPython/data/simpleruledata.txt")


if __name__ == "__main__":
    scrapdata()
```



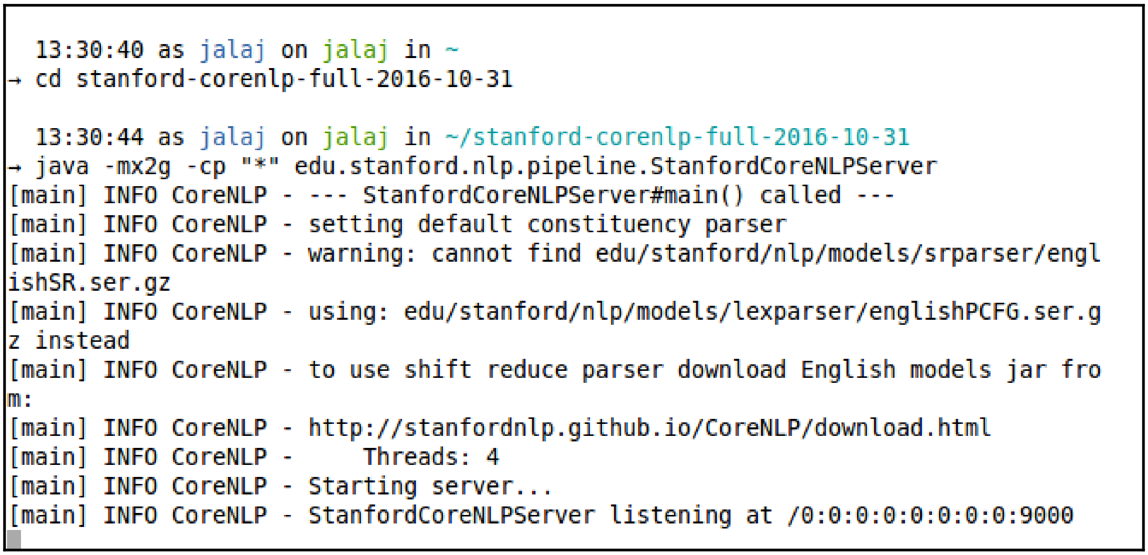
```
import re

inputstring = "Our meeting will be at 5pm tomorrow."
# inputstring = "Our meeting will be schedule at 11am tomorrow."

findpattern_am = re.search(r'\b([1-9]|0[1-9]|1[0-2]{1,2})(am)\b',
                           inputstring, re.M | re.I)
findpattern_pm = re.search(r'\b([1-9]|0[1-9]|1[0-2]{1,2})(pm)\b',
                           inputstring, re.M | re.I)

if findpattern_am:
    print findpattern_am.group()
    print re.sub(r'\b([1-9]|0[1-9]|1[0-2]{1,2})(am)\b', r'\1 a.m.', inputstring)
elif findpattern_pm:
    print findpattern_pm.group()
    print re.sub(r'\b([1-9]|0[1-9]|1[0-2]{1,2})(pm)\b', r'\1 p.m.', inputstring)
else:
    print "Not matched...!"
```



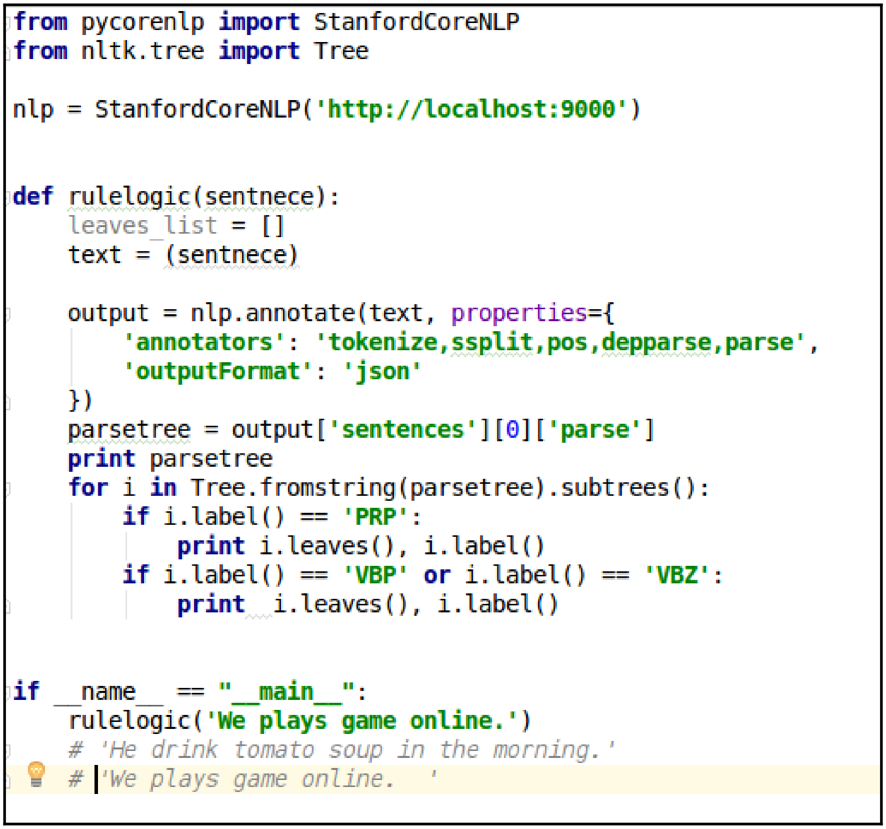
```
Our meeting will be at 5 p.m. tomorrow.
```



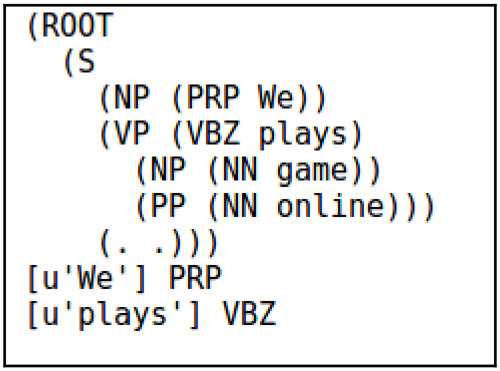
```
(ROOT
  (S
    (NP (PRP He))
    (VP (VBP drink)
      (NP
        (NP (NN tomato) (NN soup))
        (PP (IN in)
          (NP (DT the) (NN morning)))))
    (. .)))

(ROOT
  (S
    (NP (PRP She))
    (VP (VBP know)
      (NP (NN cooking)))
    (. .)))

(ROOT
  (S
    (NP (PRP we))
    (VP (VBZ plays)
      (NP (NN game))
      (PP (NN online)))
    (. .)))
```



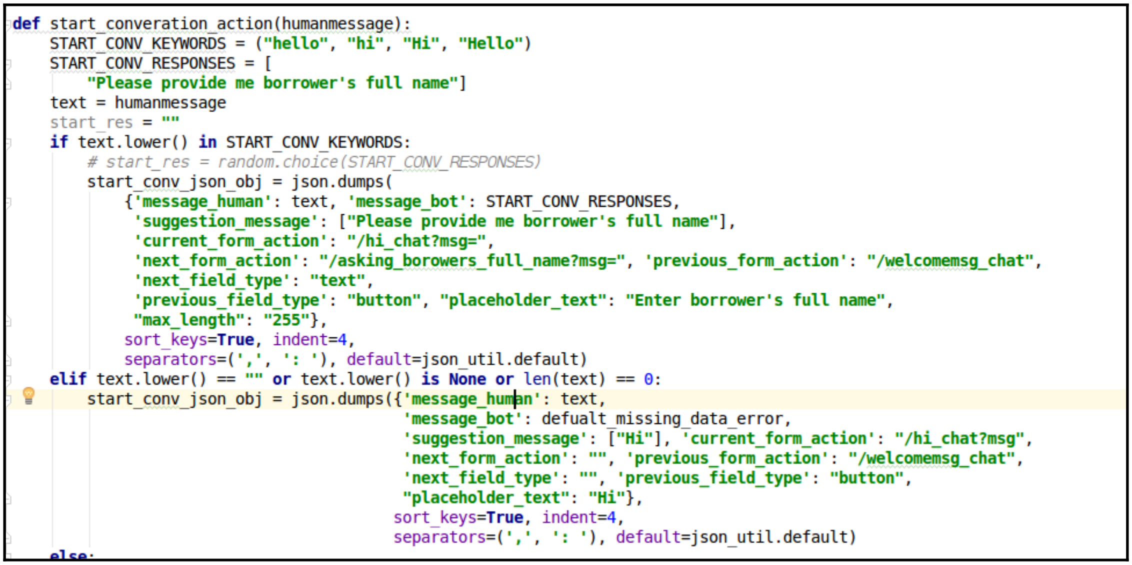
```
  13:30:40 as jalaj on jalaj in ~
→ cd stanford-corenlp-full-2016-10-31

  13:30:44 as jalaj on jalaj in ~/stanford-corenlp-full-2016-10-31
→ java -mx2g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer
[main] INFO CoreNLP - --- StanfordCoreNLPServer#main() called ---
[main] INFO CoreNLP - setting default constituency parser
[main] INFO CoreNLP - warning: cannot find edu/stanford/nlp/models/srparser/engl
ishSR.ser.gz
[main] INFO CoreNLP - using: edu/stanford/nlp/models/lexparser/englishPCFG.ser.g
z instead
[main] INFO CoreNLP - to use shift reduce parser download English models jar fro
m:
[main] INFO CoreNLP - http://stanfordnlp.github.io/CoreNLP/download.html
[main] INFO CoreNLP -     Threads: 4
[main] INFO CoreNLP - Starting server...
[main] INFO CoreNLP - StanfordCoreNLPServer listening at /0:0:0:0:0:0:0:0:9000
```



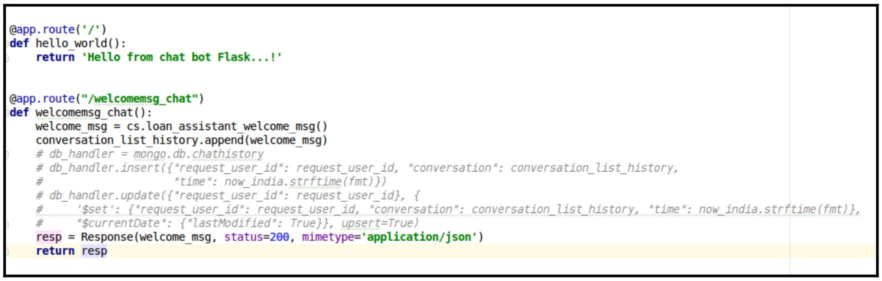
```
from pycorenlp import StanfordCoreNLP
from nltk.tree import Tree

nlp = StanfordCoreNLP('http://localhost:9000')


def rulelogic(sentnece):
    leaves_list = []
    text = (sentnece)

    output = nlp.annotate(text, properties={
        'annotators': 'tokenize,ssplit,pos,depparse,parse',
        'outputFormat': 'json'
    })
    parsetree = output['sentences'][0]['parse']
    print parsetree
    for i in Tree.fromstring(parsetree).subtrees():
        if i.label() == 'PRP':
            print i.leaves(), i.label()
        if i.label() == 'VBP' or i.label() == 'VBZ':
            print  i.leaves(), i.label()


if __name__ == "__main__":
    rulelogic('We plays game online.')
    # 'He drink tomato soup in the morning.'
    # 'We plays game online.  '
```



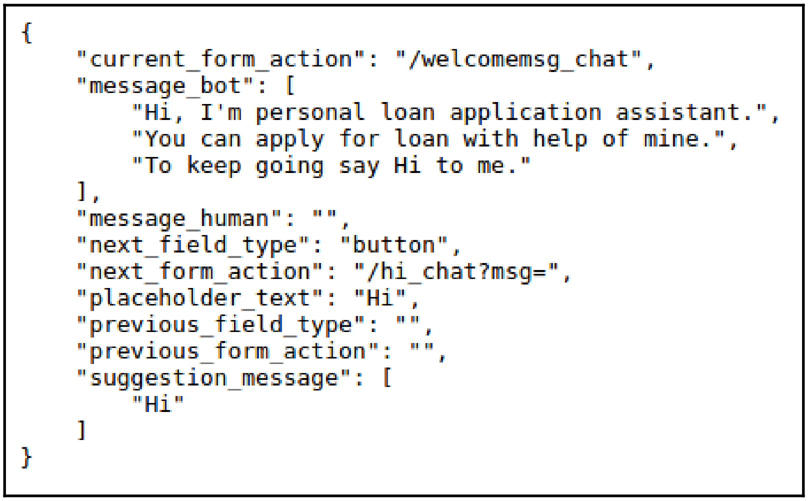
```
(ROOT
  (S
    (NP (PRP We))
    (VP (VBZ plays)
      (NP (NN game))
      (PP (NN online)))
    (. .)))
[u'We'] PRP
[u'plays'] VBZ
```



```python
def start_converation_action(humanmessage):
    START_CONV_KEYWORDS = ("hello", "hi", "Hi", "Hello")
    START_CONV_RESPONSES = [
        "Please provide me borrower's full name"]
    text = humanmessage
    start_res = ""
    if text.lower() in START_CONV_KEYWORDS:
        # start_res = random.choice(START_CONV_RESPONSES)
        start_conv_json_obj = json.dumps(
            {'message_human': text, 'message_bot': START_CONV_RESPONSES,
             'suggestion_message': ["Please provide me borrower's full name"],
             'current_form_action': "/hi_chat?msg=",
             'next_form_action': "/asking_borowers_full_name?msg=", 'previous_form_action': "/welcomemsg_chat",
             'next_field_type': "text",
             'previous_field_type': "button", "placeholder_text": "Enter borrower's full name",
             "max_length": "255"},
            sort_keys=True, indent=4,
            separators=(',', ': '), default=json_util.default)
    elif text.lower() == "" or text.lower() is None or len(text) == 0:
        start_conv_json_obj = json.dumps({'message_human': text,
                                          'message_bot': defualt_missing_data_error,
                                          'suggestion_message': ["Hi"], 'current_form_action': "/hi_chat?msg",
                                          'next_form_action': "", 'previous_form_action': "/welcomemsg_chat",
                                          'next_field_type': "", 'previous_field_type': "button",
                                          "placeholder_text": "Hi"},
                                         sort_keys=True, indent=4,
                                         separators=(',', ': '), default=json_util.default)
    else:
```

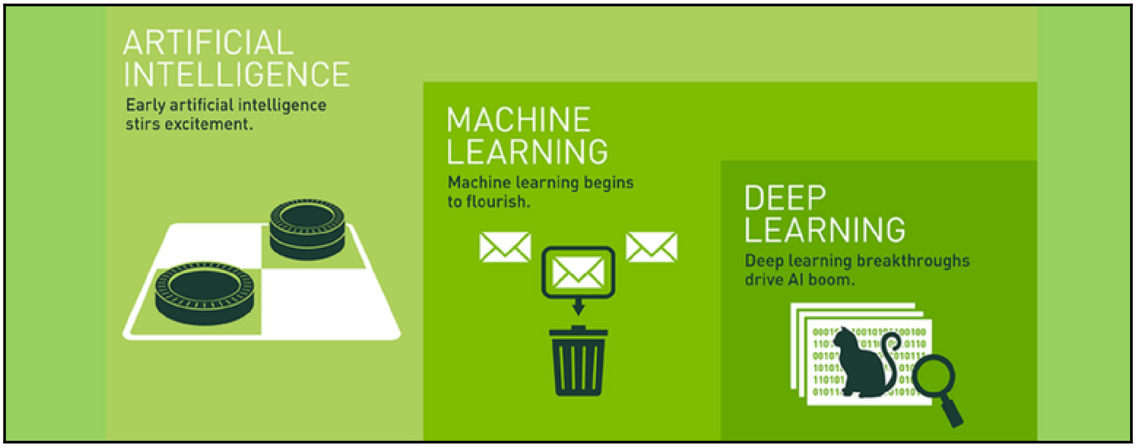


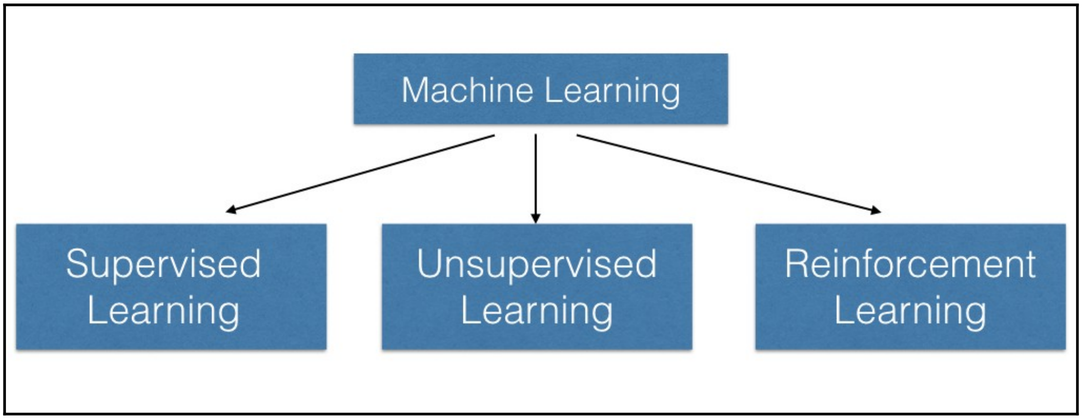
```python
@app.route('/')
def hello_world():
    return 'Hello from chat bot Flask...!'


@app.route("/welcomemsg_chat")
def welcomemsg_chat():
    welcome_msg = cs.loan_assistant_welcome_msg()
    conversation_list_history.append(welcome_msg)
    # db_handler = mongo.db.chathistory
    # db_handler.insert({"request_user_id": request_user_id, "conversation": conversation_list_history,
    #                    "time": now_india.strftime(fmt)})
    # db_handler.update({"request_user_id": request_user_id}, {
    #     '$set': {"request_user_id": request_user_id, "conversation": conversation_list_history, "time": now_india.strftime(fmt)},
    #     "$currentDate": {"lastModified": True}}, upsert=True)
    resp = Response(welcome_msg, status=200, mimetype='application/json')
    return resp
```

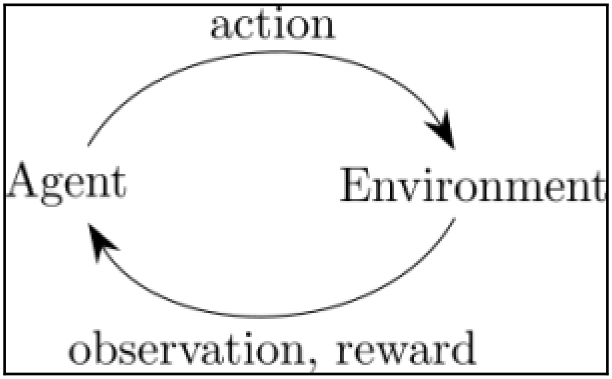


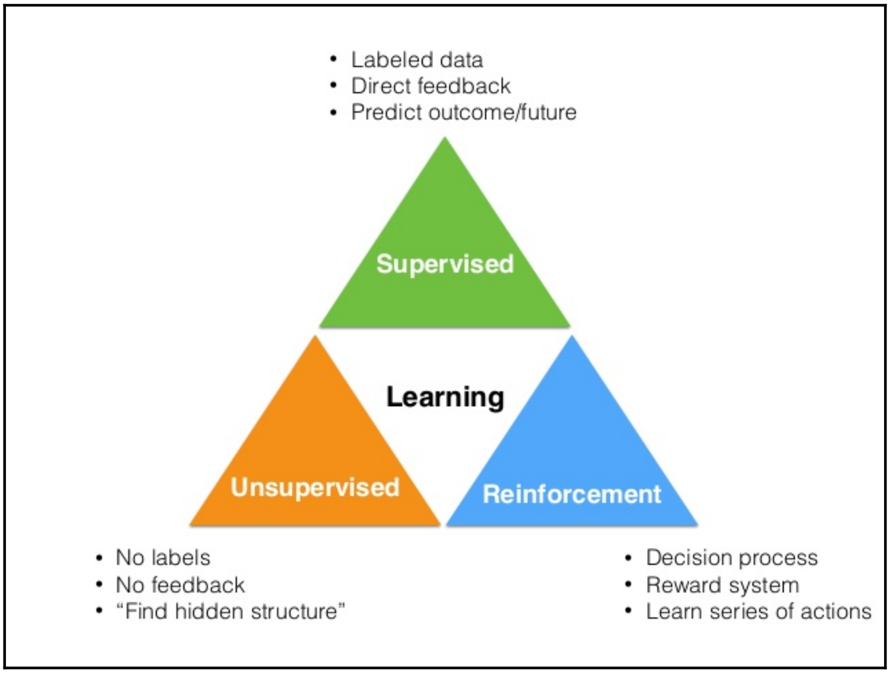
```json
{
    "current_form_action": "/welcomemsg_chat",
    "message_bot": [
        "Hi, I'm personal loan application assistant.",
        "You can apply for loan with help of mine.",
        "To keep going say Hi to me."
    ],
    "message_human": "",
    "next_field_type": "button",
    "next_form_action": "/hi_chat?msg=",
    "placeholder_text": "Hi",
    "previous_field_type": "",
    "previous_form_action": "",
    "suggestion_message": [
        "Hi"
    ]
}
```

Chapter 8: Machine Learning for NLP Problems


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.



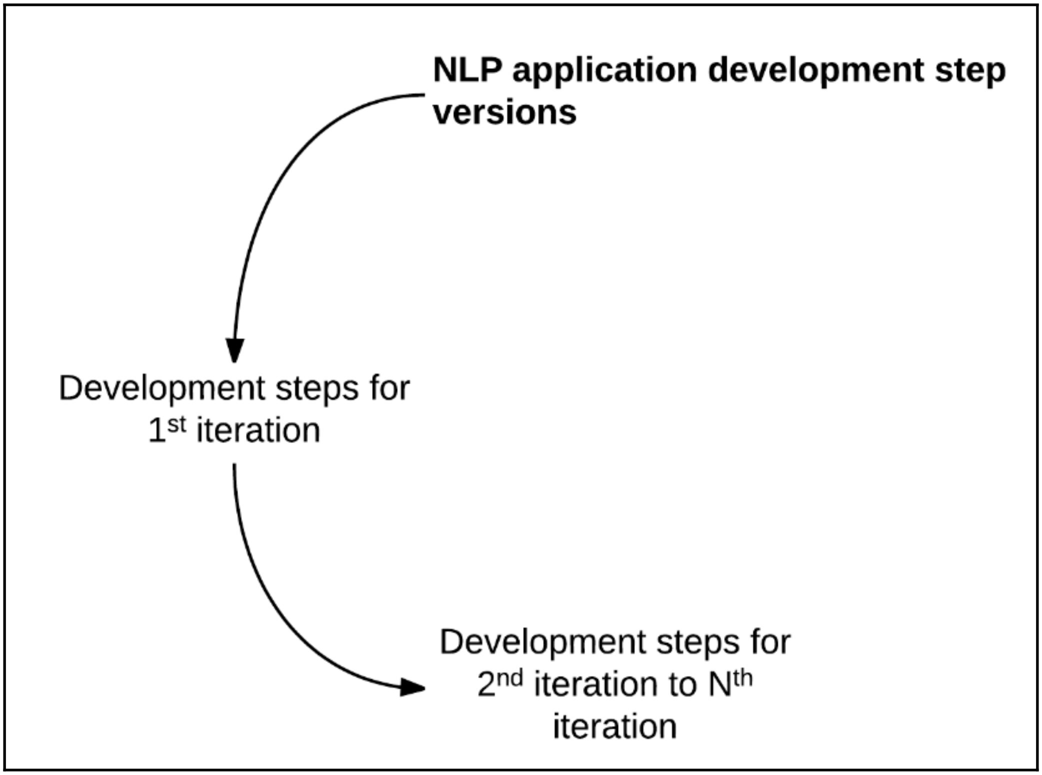


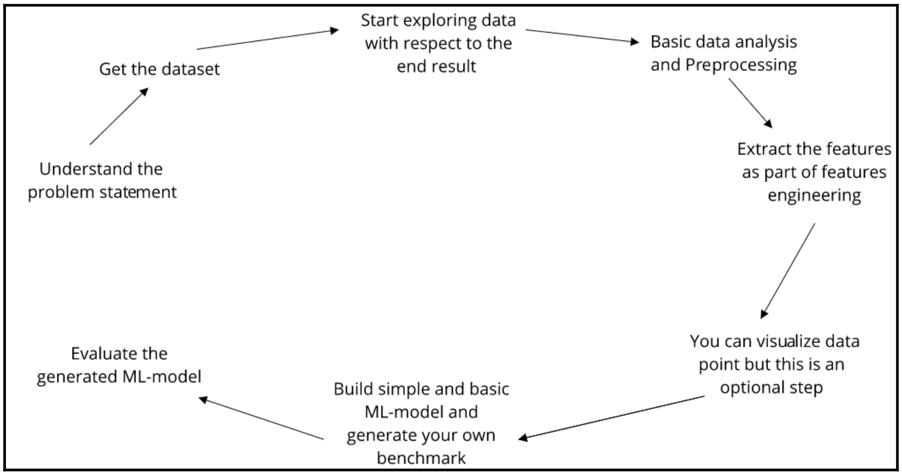


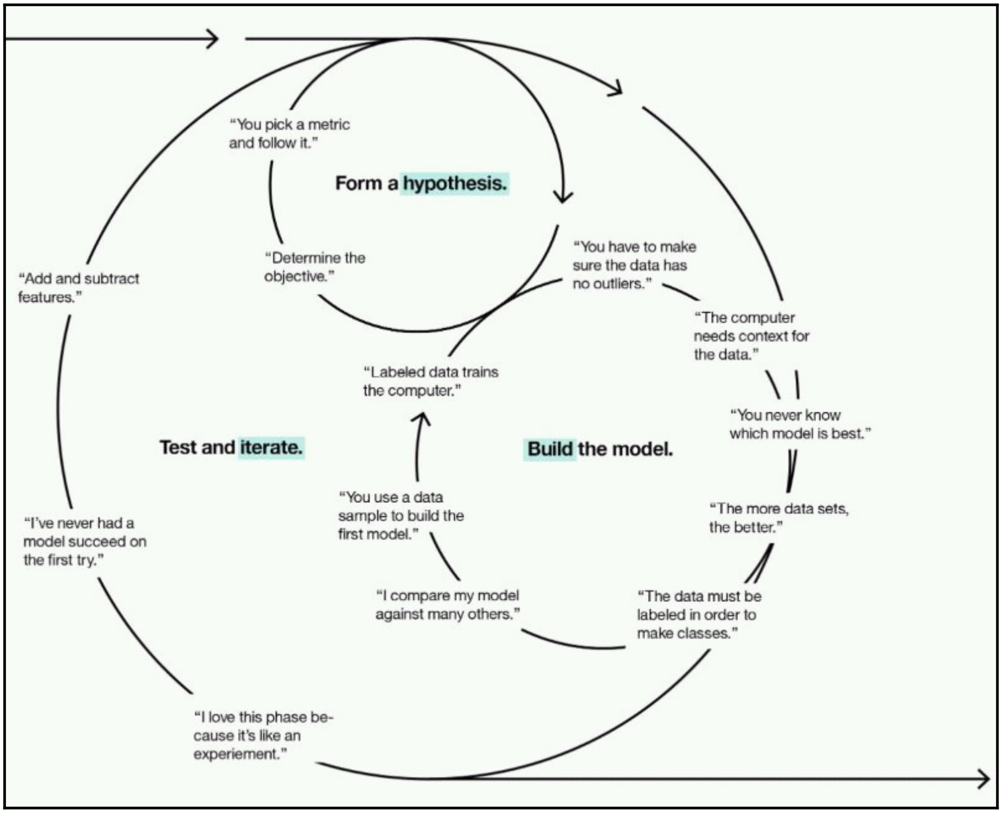
- Labeled data
- Direct feedback
- Predict outcome/future

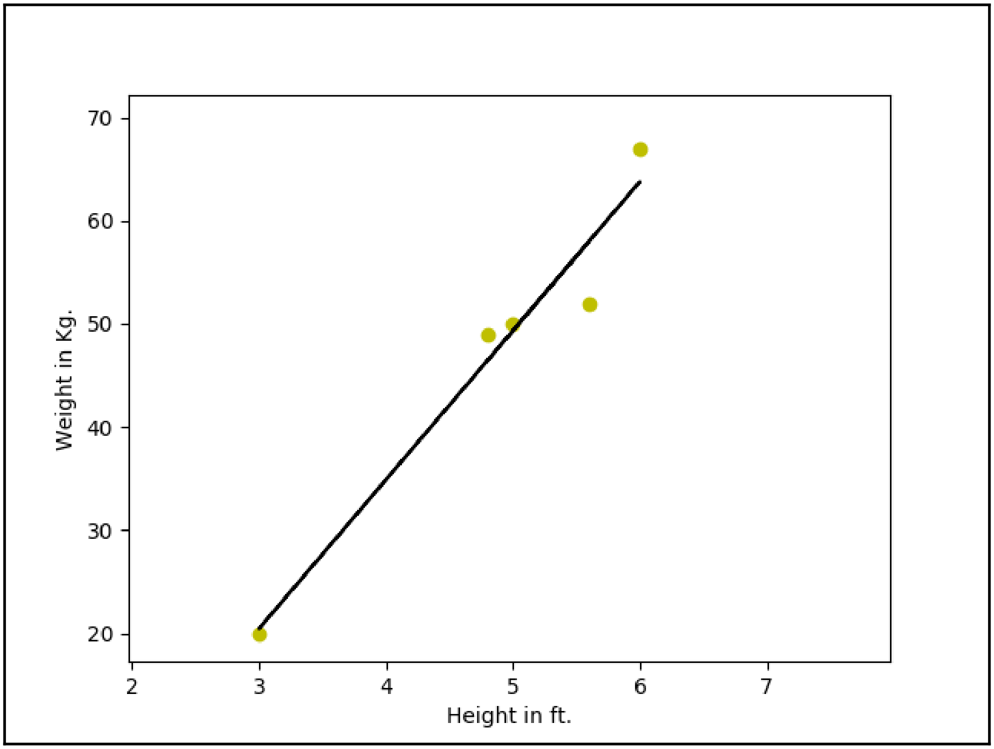
- No labels
- No feedback
- "Find hidden structure"

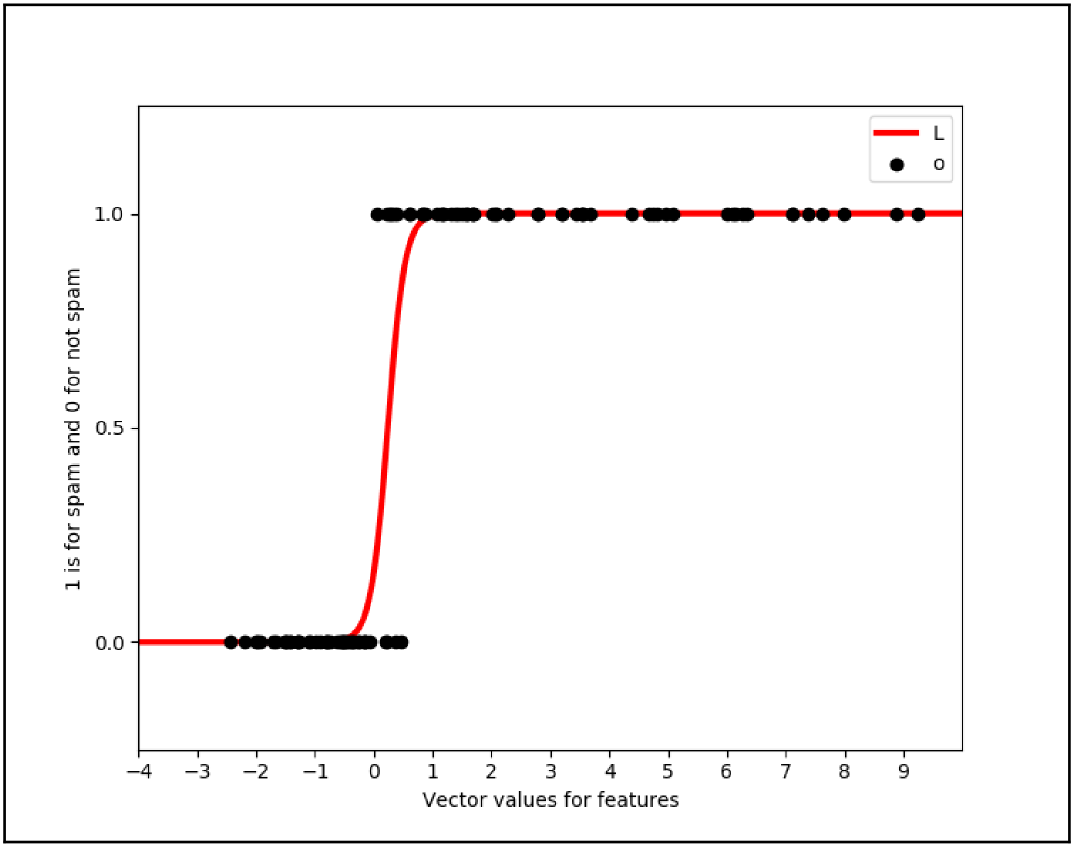
- Decision process
- Reward system
- Learn series of actions

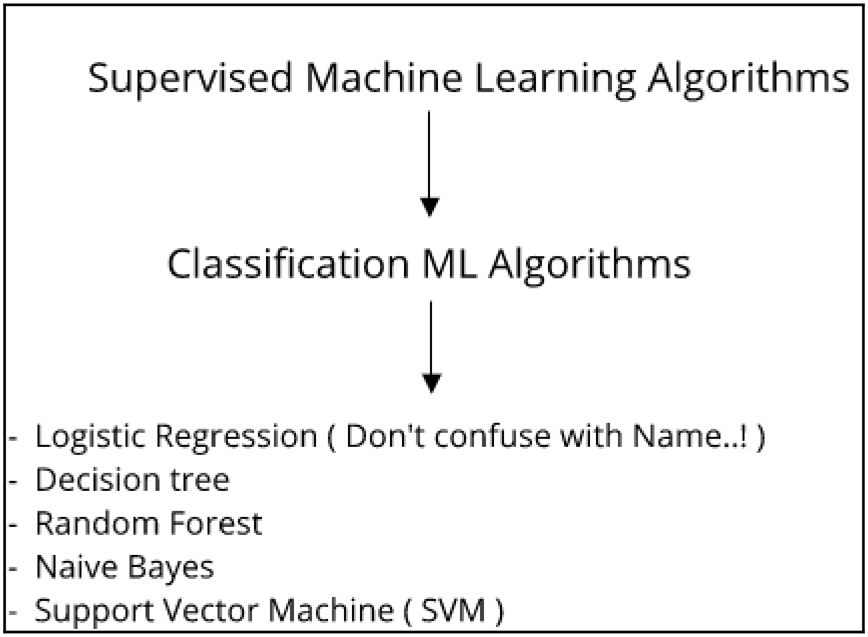

NLP application development step versions
Development steps for 1st iteration
Development steps for 2nd iteration to Nth iteration


Understand the problem statement
Get the dataset
Start exploring data with respect to the end result
Basic data analysis and Preprocessing
Extract the features as part of features engineering
You can visualize data point but this is an optional step
Build simple and basic ML-model and generate your own benchmark
Evaluate the generated ML-model


Form a hypothesis.
"You pick a metric and follow it."
"Determine the objective."
"You have to make sure the data has no outliers."
"The computer needs context for the data."
"You never know which model is best."
"The more data sets, the better."
"The data must be labeled in order to make classes."
Build the model.
"Labeled data trains the computer."
"You use a data sample to build the first model."
"I compare my model against many others."
Test and iterate.
"Add and subtract features."
"I've never had a model succeed on the first try."
"I love this phase be-cause it's like an experiement."


70
60
50
40
30
20
Weight in Kg.
2
3
4
5
6
7
Height in ft.


L
o
1.0
0.5
0.0
1 is for spam and 0 for not spam
−4
−3
−2
−1
0
1
2
3
4
5
6
7
8
9
Vector values for features



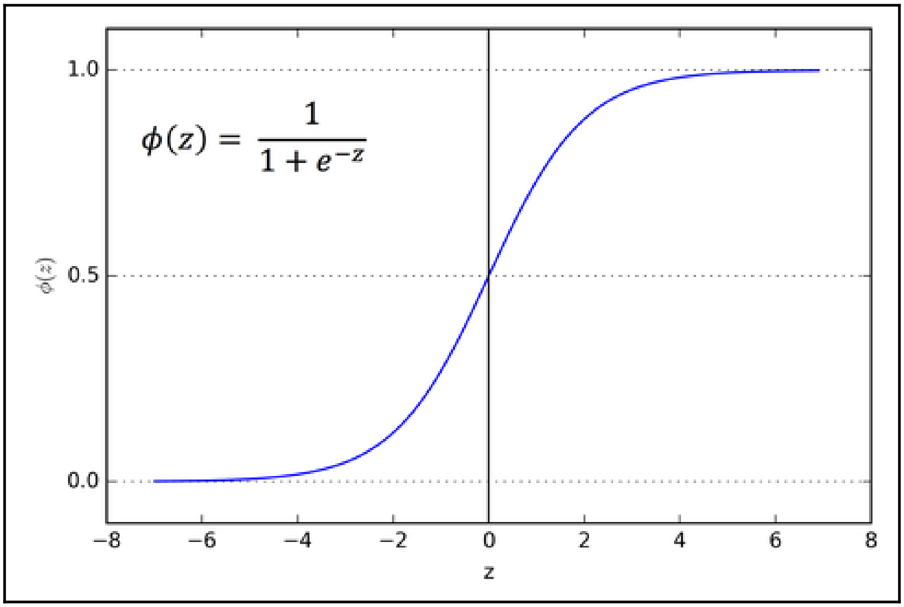
Supervised Machine Learning Algorithms

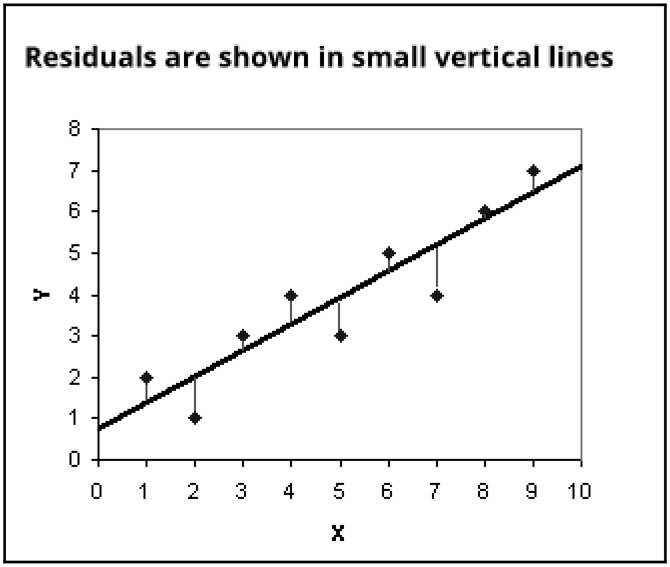
↓

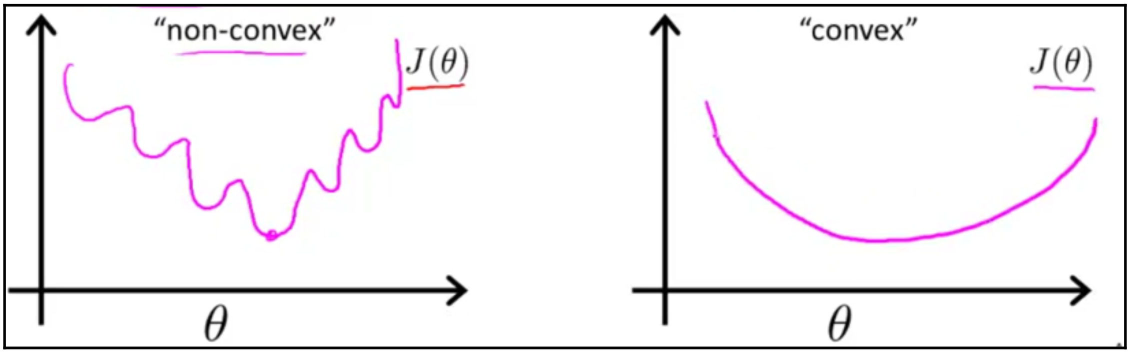
Classification ML Algorithms

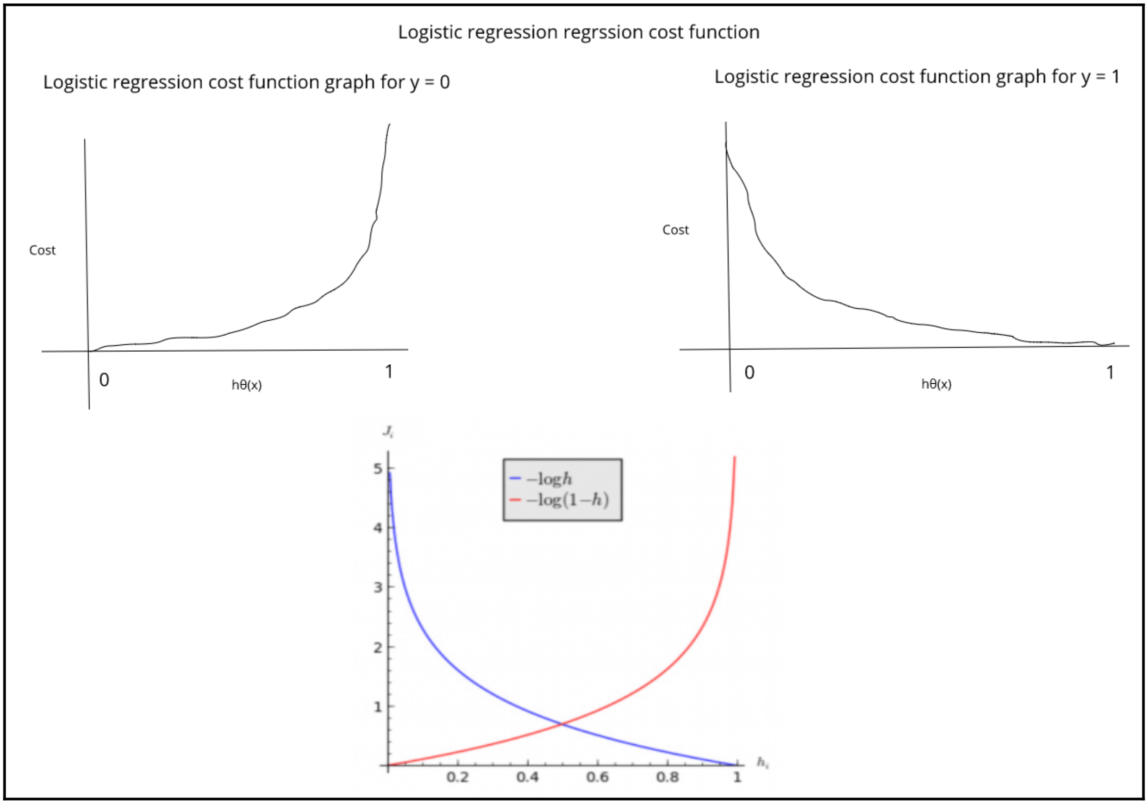
↓

- Logistic Regression (Don't confuse with Name..!)
- Decision tree
- Random Forest
- Naive Bayes
- Support Vector Machine (SVM)




Residuals are shown in small vertical lines
Y
0
1
2
3
4
5
6
7
8
X
0
1
2
3
4
5
6
7
8
9
10

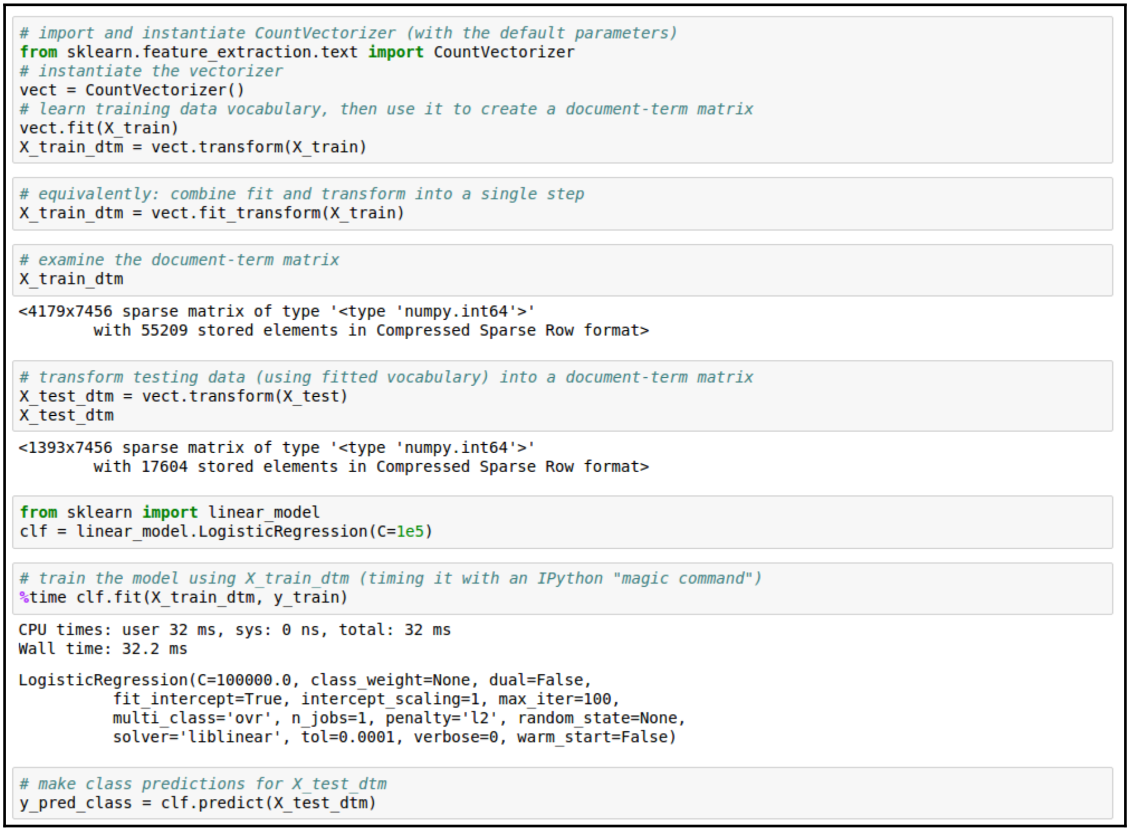

"non-convex"
$J(\theta)$
θ
"convex"
$J(\theta)$
θ



Logistic regression regrssion cost function

Logistic regression cost function graph for y = 0

Logistic regression cost function graph for y = 1



```
# import and instantiate CountVectorizer (with the default parameters)
from sklearn.feature_extraction.text import CountVectorizer
# instantiate the vectorizer
vect = CountVectorizer()
# learn training data vocabulary, then use it to create a document-term matrix
vect.fit(X_train)
X_train_dtm = vect.transform(X_train)
```

```
# equivalently: combine fit and transform into a single step
X_train_dtm = vect.fit_transform(X_train)
```

```
# examine the document-term matrix
X_train_dtm
```

```
<4179x7456 sparse matrix of type '<type 'numpy.int64'>'
	with 55209 stored elements in Compressed Sparse Row format>
```

```
# transform testing data (using fitted vocabulary) into a document-term matrix
X_test_dtm = vect.transform(X_test)
X_test_dtm
```

```
<1393x7456 sparse matrix of type '<type 'numpy.int64'>'
	with 17604 stored elements in Compressed Sparse Row format>
```

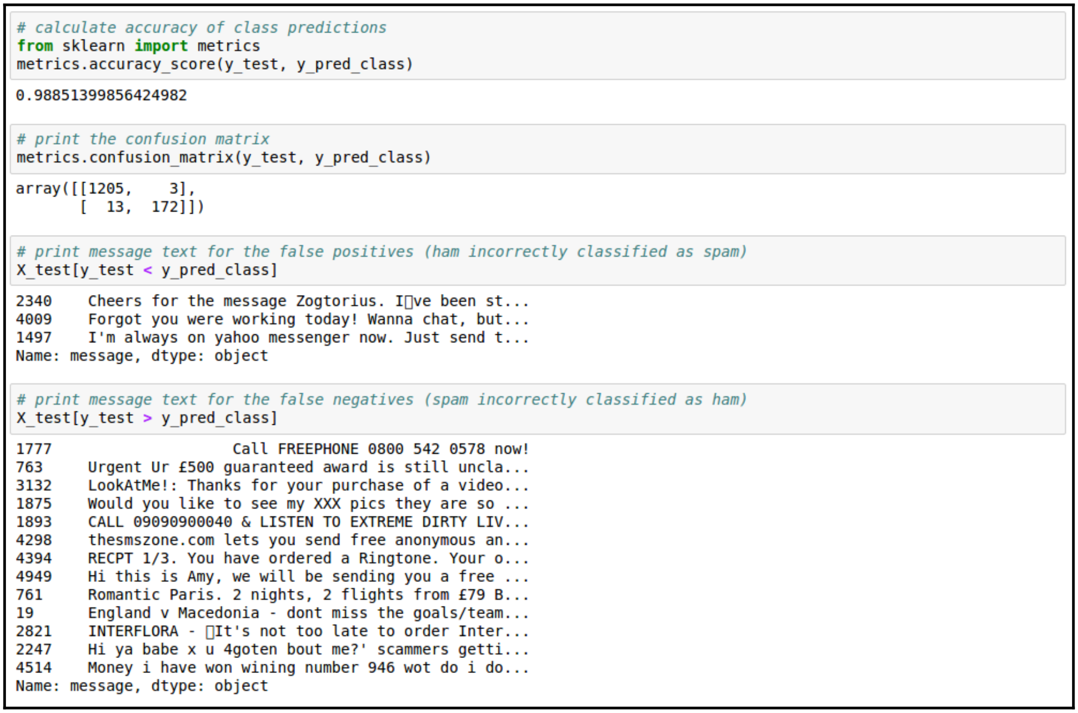
```
from sklearn import linear_model
clf = linear_model.LogisticRegression(C=1e5)
```

```
# train the model using X_train_dtm (timing it with an IPython "magic command")
%time clf.fit(X_train_dtm, y_train)
```

```
CPU times: user 32 ms, sys: 0 ns, total: 32 ms
Wall time: 32.2 ms
```

```
LogisticRegression(C=100000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

```
# make class predictions for X_test_dtm
y_pred_class = clf.predict(X_test_dtm)
```



```
# calculate accuracy of class predictions
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred_class)
```

```
0.988513998564249282
```

```
# print the confusion matrix
metrics.confusion_matrix(y_test, y_pred_class)
```

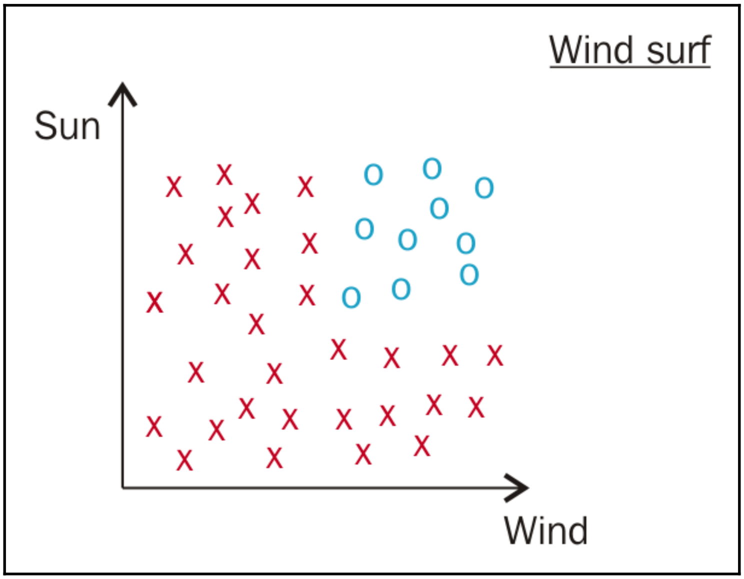
```
array([[1205,    3],
       [  13,  172]])
```

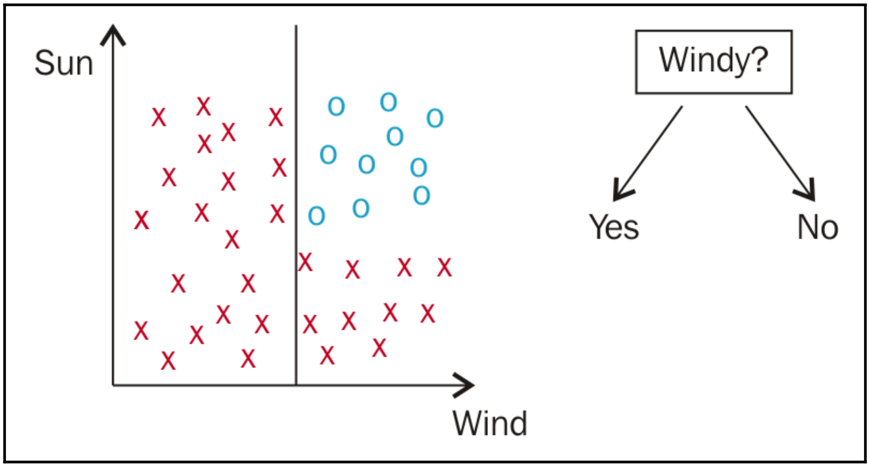
```
# print message text for the false positives (ham incorrectly classified as spam)
X_test[y_test < y_pred_class]
```

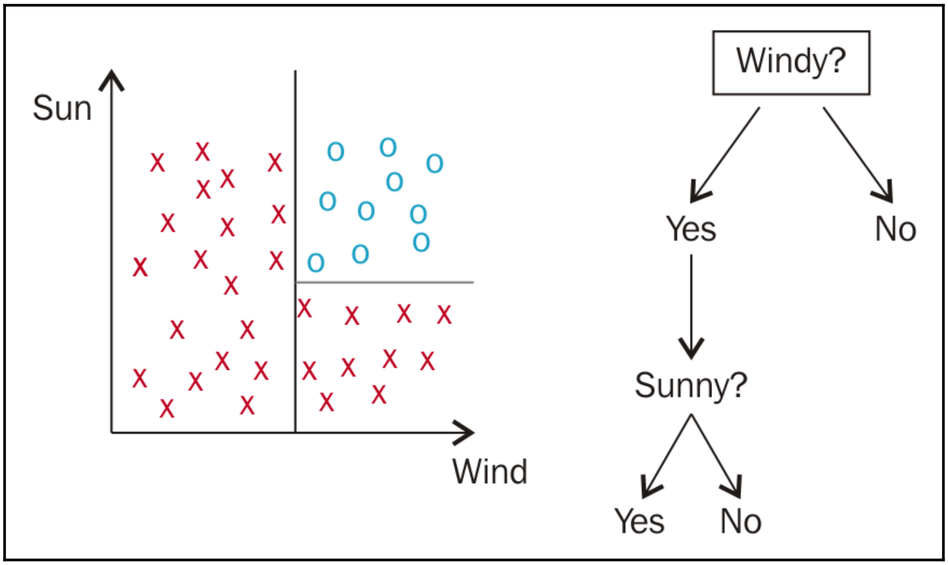
```
2340    Cheers for the message Zogtorius. I□ve been st...
4009    Forgot you were working today! Wanna chat, but...
1497    I'm always on yahoo messenger now. Just send t...
Name: message, dtype: object
```

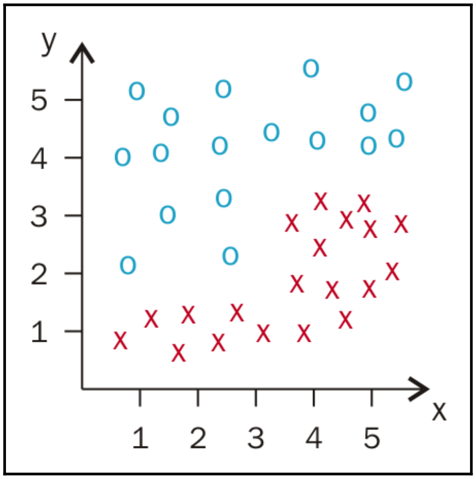
```
# print message text for the false negatives (spam incorrectly classified as ham)
X_test[y_test > y_pred_class]
```

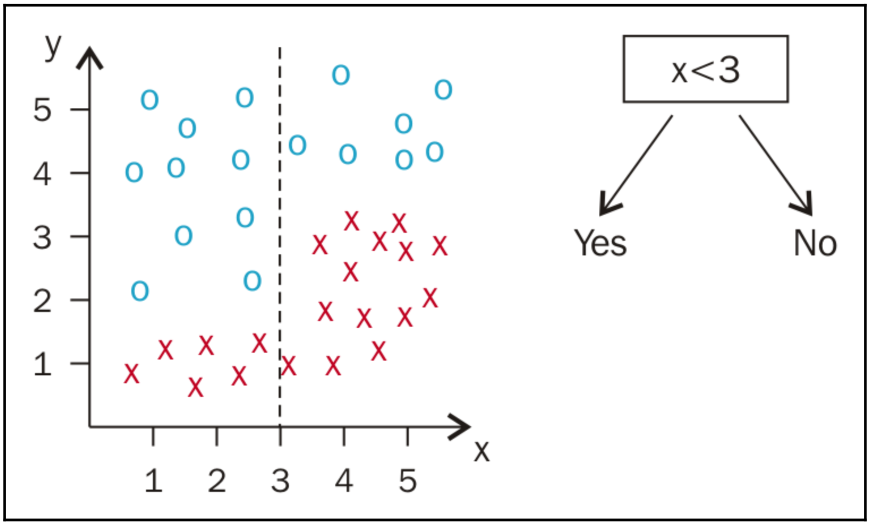
```
1777                        Call FREEPHONE 0800 542 0578 now!
763     Urgent Ur £500 guaranteed award is still uncla...
3132    LookAtMe!: Thanks for your purchase of a video...
1875    Would you like to see my XXX pics they are so ...
1893    CALL 09090900040 & LISTEN TO EXTREME DIRTY LIV...
4298    thesmszone.com lets you send free anonymous an...
4394    RECPT 1/3. You have ordered a Ringtone. Your o...
4949    Hi this is Amy, we will be sending you a free ...
761     Romantic Paris. 2 nights, 2 flights from £79 B...
19      England v Macedonia - dont miss the goals/team...
2821    INTERFLORA - □It's not too late to order Inter...
2247    Hi ya babe x u 4goten bout me?' scammers getti...
4514    Money i have won wining number 946 wot do i do...
Name: message, dtype: object
```

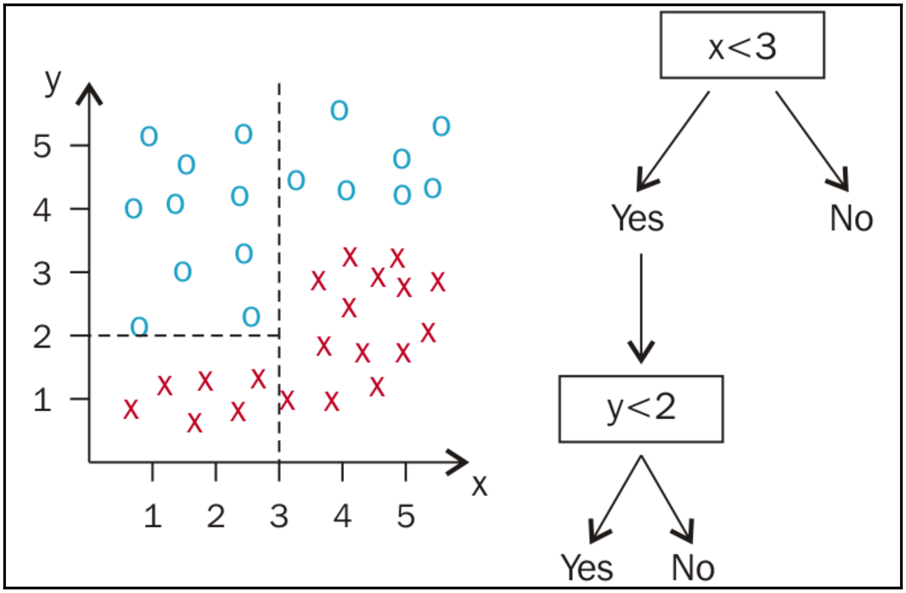


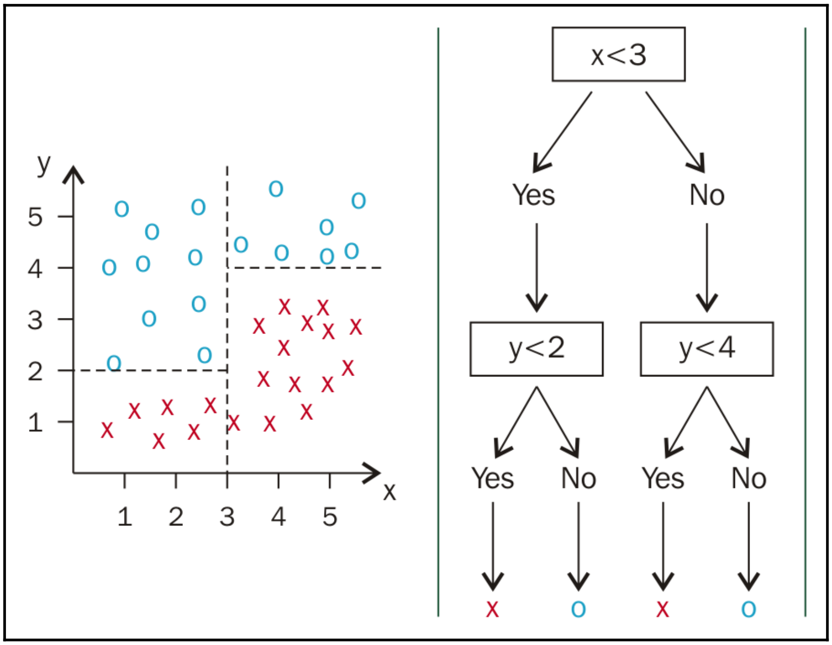

Sun
Wind
Windy?
Yes
No

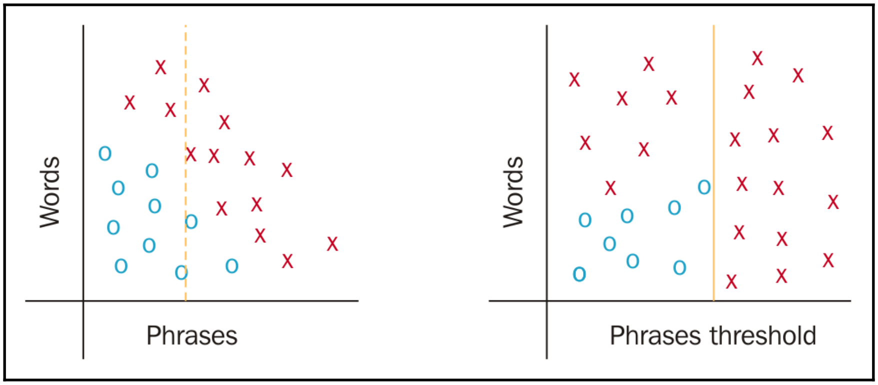

Sun
Wind
Windy?
Yes
No
Sunny?
Yes
No

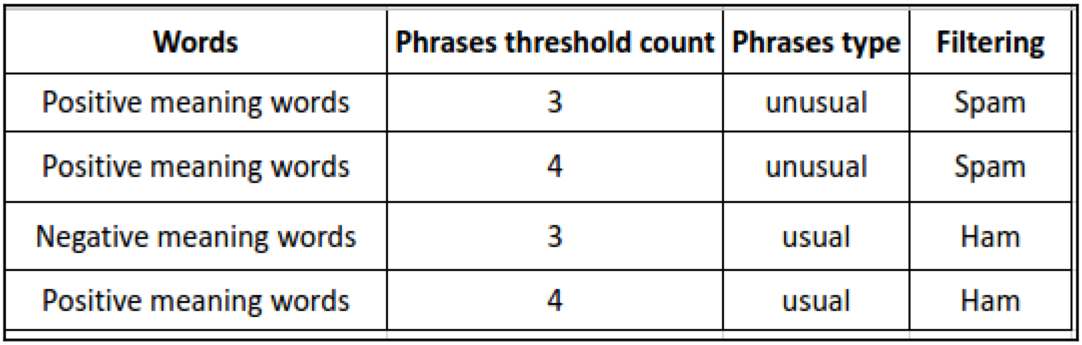

y
5
4
3
2
1
1
2
3
4
5
x


y
5
4
3
2
1
1
2
3
4
5
x
x<3
Yes
No

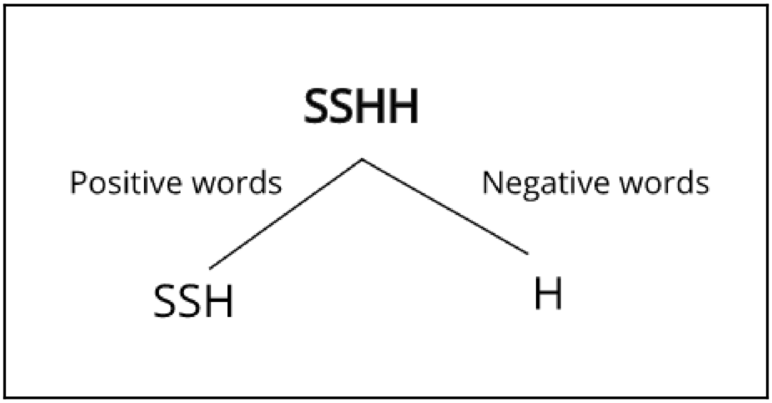

y
5
4
3
2
1
1
2
3
4
5
x
x<3
Yes
No
y<2
Yes
No

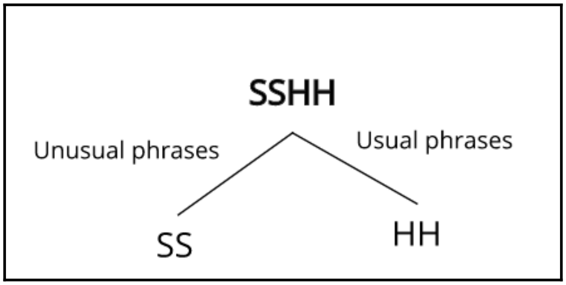

y
5
4
3
2
1
1
2
3
4
5
x
x<3
Yes
No
y<2
y<4
Yes
No
Yes
No
x
o
x
o

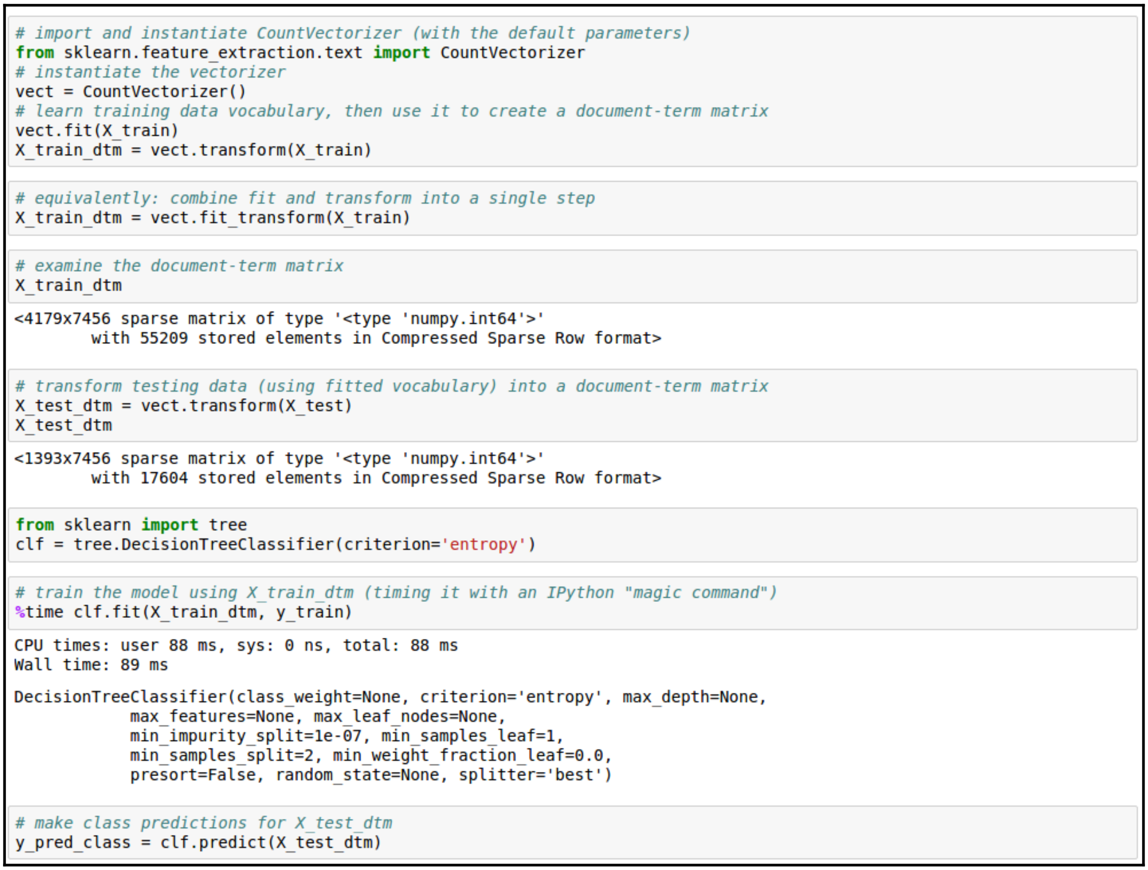




Words	Phrases threshold count	Phrases type	Filtering
Positive meaning words	3	unusual	Spam
Positive meaning words	4	unusual	Spam
Negative meaning words	3	usual	Ham
Positive meaning words	4	usual	Ham







```
# import and instantiate CountVectorizer (with the default parameters)
from sklearn.feature_extraction.text import CountVectorizer
# instantiate the vectorizer
vect = CountVectorizer()
# learn training data vocabulary, then use it to create a document-term matrix
vect.fit(X_train)
X_train_dtm = vect.transform(X_train)
```

```
# equivalently: combine fit and transform into a single step
X_train_dtm = vect.fit_transform(X_train)
```

```
# examine the document-term matrix
X_train_dtm
```

```
<4179x7456 sparse matrix of type '<type 'numpy.int64'>'
	with 55209 stored elements in Compressed Sparse Row format>
```

```
# transform testing data (using fitted vocabulary) into a document-term matrix
X_test_dtm = vect.transform(X_test)
X_test_dtm
```

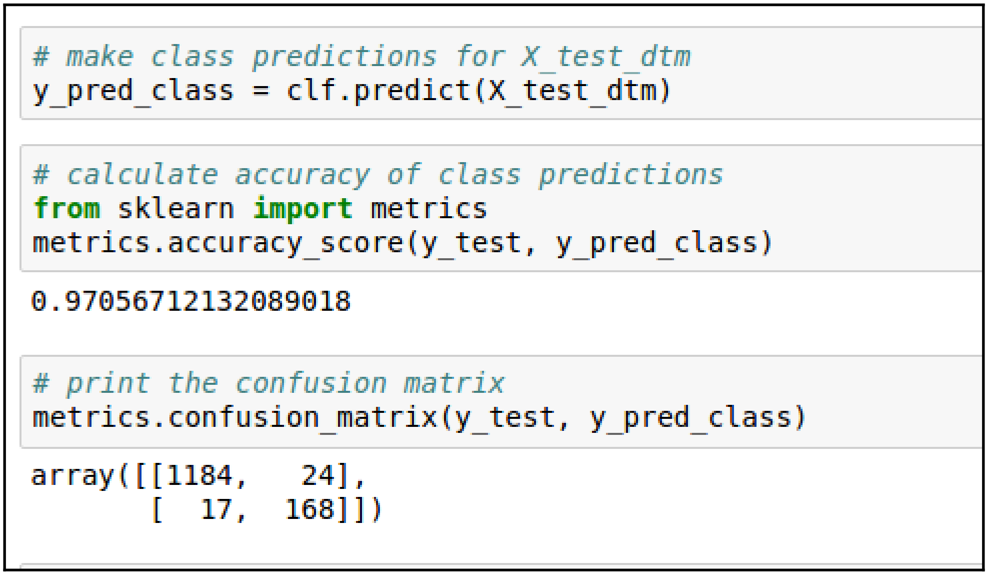
```
<1393x7456 sparse matrix of type '<type 'numpy.int64'>'
	with 17604 stored elements in Compressed Sparse Row format>
```

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='entropy')
```

```
# train the model using X_train_dtm (timing it with an IPython "magic command")
%time clf.fit(X_train_dtm, y_train)
```

```
CPU times: user 88 ms, sys: 0 ns, total: 88 ms
Wall time: 89 ms

DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

```
# make class predictions for X_test_dtm
y_pred_class = clf.predict(X_test_dtm)
```



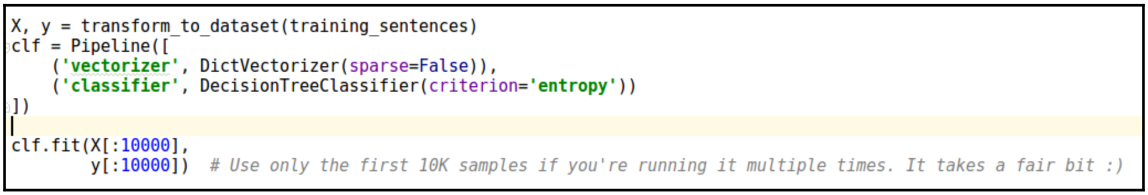
```
# make class predictions for X_test_dtm
y_pred_class = clf.predict(X_test_dtm)
```

```
# calculate accuracy of class predictions
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred_class)
```

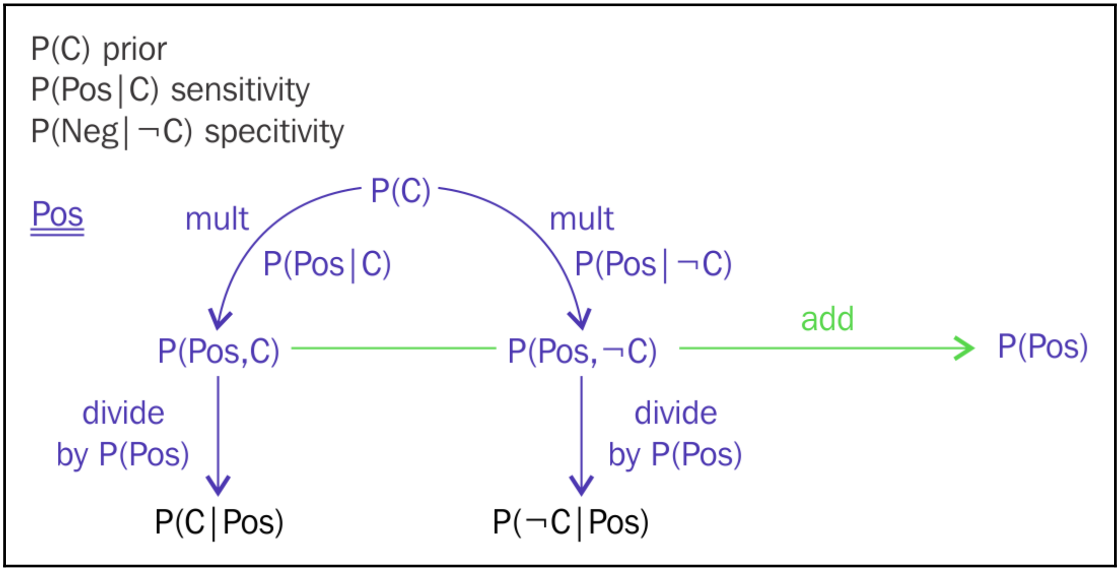
```
0.97056712132089018
```

```
# print the confusion matrix
metrics.confusion_matrix(y_test, y_pred_class)
```

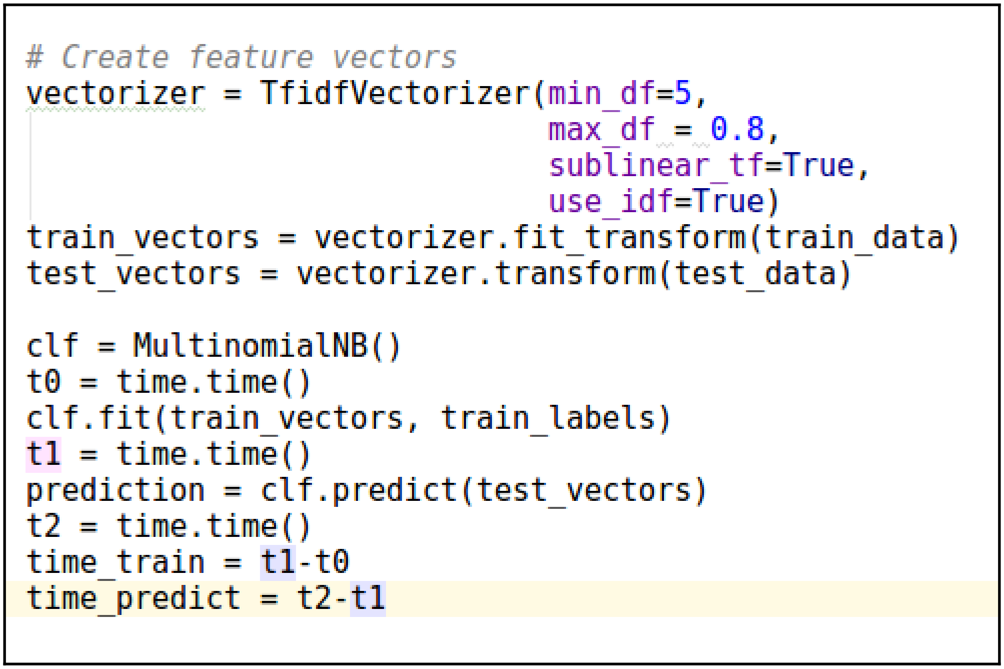
```
array([[1184,   24],
       [  17,  168]])
```



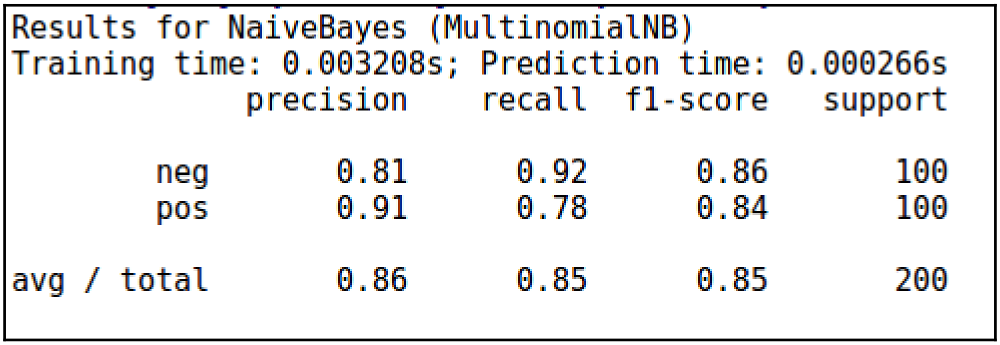
```
X, y = transform_to_dataset(training_sentences)
clf = Pipeline([
    ('vectorizer', DictVectorizer(sparse=False)),
    ('classifier', DecisionTreeClassifier(criterion='entropy'))
])

clf.fit(X[:10000],
        y[:10000])  # Use only the first 10K samples if you're running it multiple times. It takes a fair bit :)
```



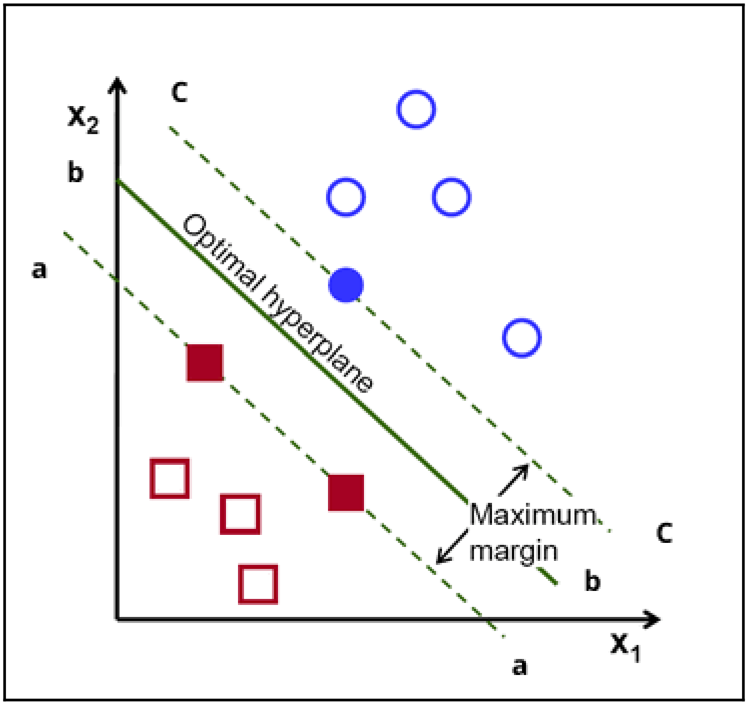
P(C) prior
P(Pos|C) sensitivity
P(Neg|¬C) specitivity

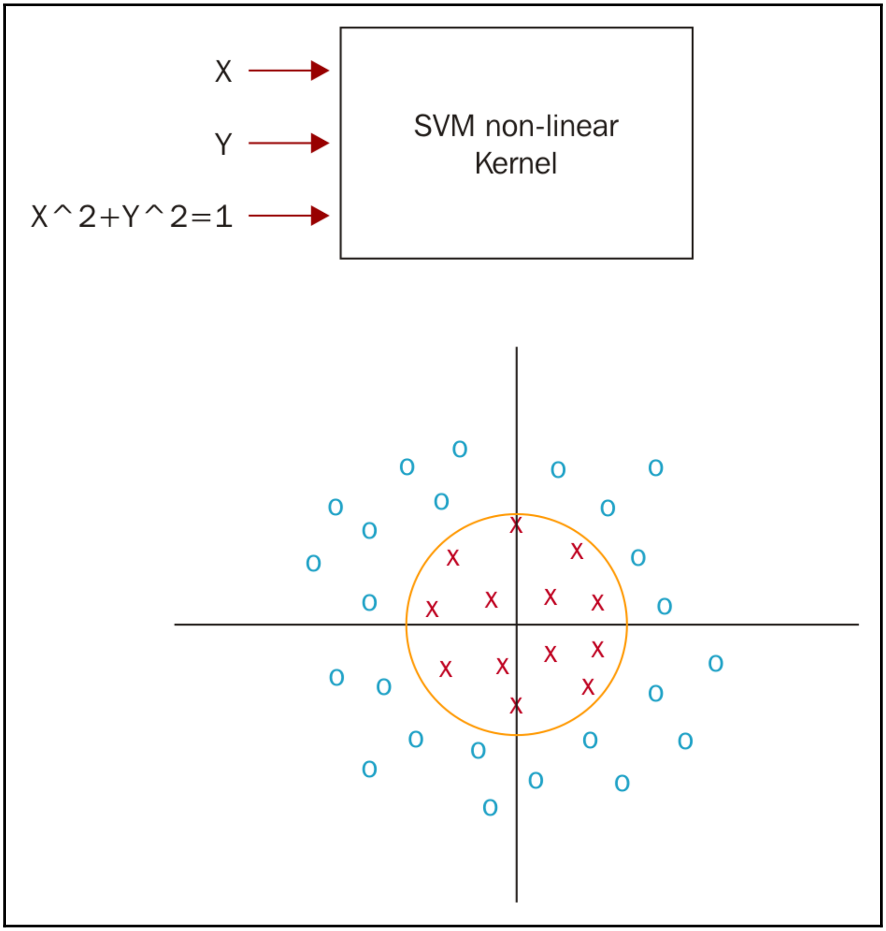


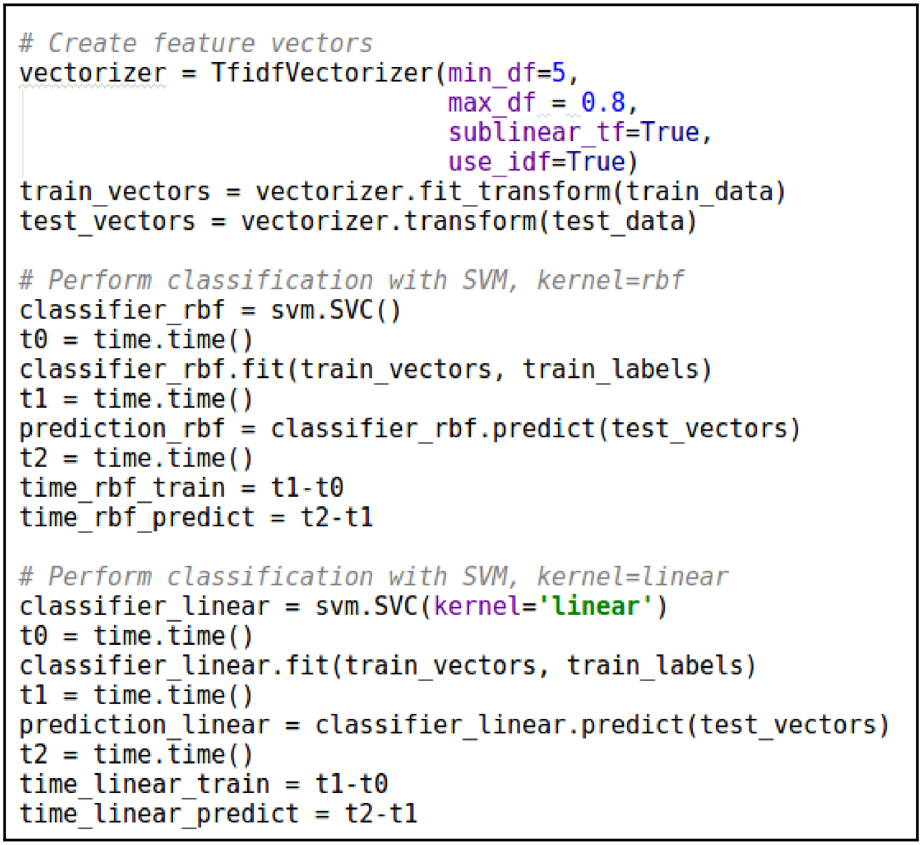
```
# Create feature vectors
vectorizer = TfidfVectorizer(min_df=5,
                             max_df = 0.8,
                             sublinear_tf=True,
                             use_idf=True)
train_vectors = vectorizer.fit_transform(train_data)
test_vectors = vectorizer.transform(test_data)

clf = MultinomialNB()
t0 = time.time()
clf.fit(train_vectors, train_labels)
t1 = time.time()
prediction = clf.predict(test_vectors)
t2 = time.time()
time_train = t1-t0
time_predict = t2-t1
```



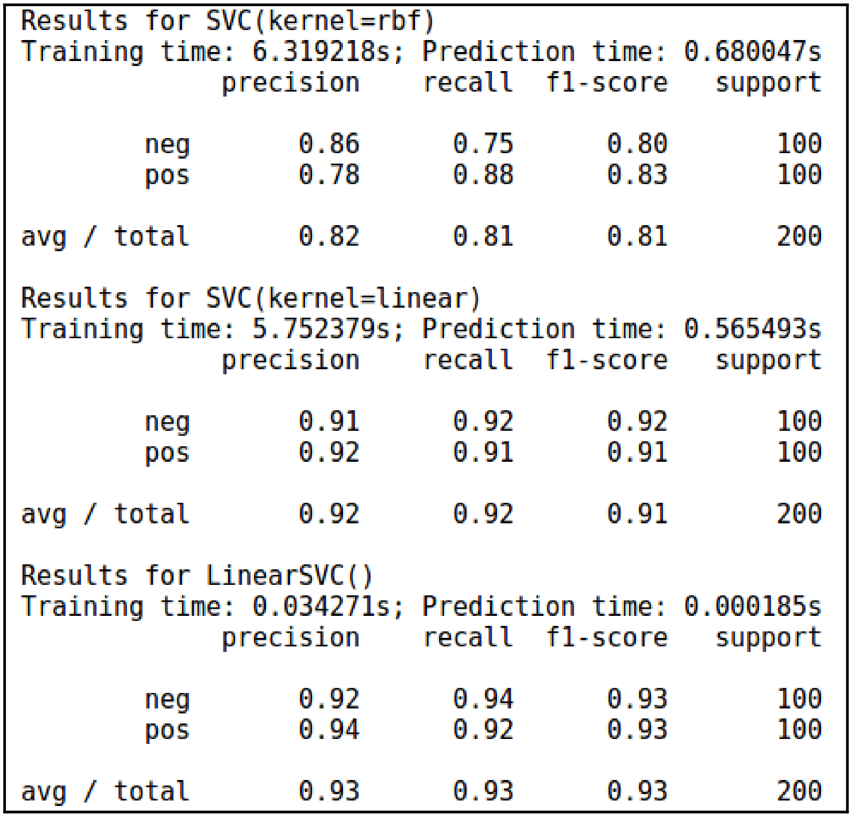
```
Results for NaiveBayes (MultinomialNB)
Training time: 0.003208s; Prediction time: 0.000266s
             precision    recall  f1-score   support

        neg       0.81      0.92      0.86       100
        pos       0.91      0.78      0.84       100

avg / total       0.86      0.85      0.85       200
```


c
x$_2$
b
a
Optimal hyperplane
Maximum
margin
c
b
x$_1$
a

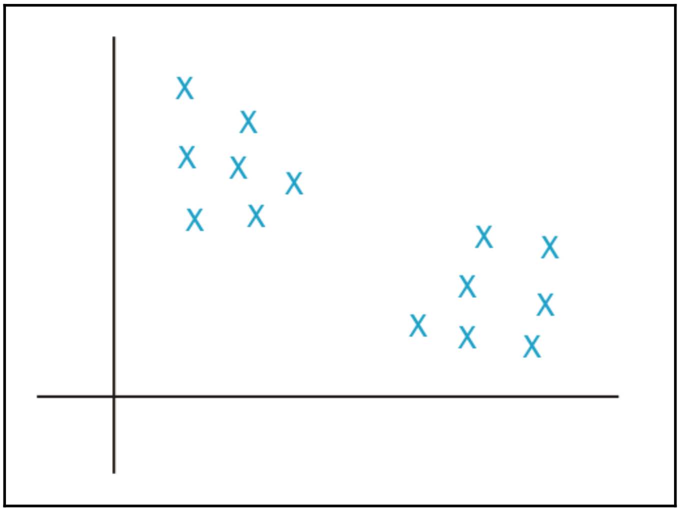

X
Y
X^2+Y^2=1
SVM non-linear Kernel

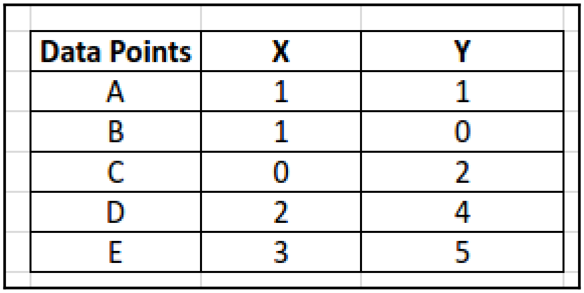


```
# Create feature vectors
vectorizer = TfidfVectorizer(min_df=5,
                             max_df = 0.8,
                             sublinear_tf=True,
                             use_idf=True)
train_vectors = vectorizer.fit_transform(train_data)
test_vectors = vectorizer.transform(test_data)

# Perform classification with SVM, kernel=rbf
classifier_rbf = svm.SVC()
t0 = time.time()
classifier_rbf.fit(train_vectors, train_labels)
t1 = time.time()
prediction_rbf = classifier_rbf.predict(test_vectors)
t2 = time.time()
time_rbf_train = t1-t0
time_rbf_predict = t2-t1

# Perform classification with SVM, kernel=linear
classifier_linear = svm.SVC(kernel='linear')
t0 = time.time()
classifier_linear.fit(train_vectors, train_labels)
t1 = time.time()
prediction_linear = classifier_linear.predict(test_vectors)
t2 = time.time()
time_linear_train = t1-t0
time_linear_predict = t2-t1
```

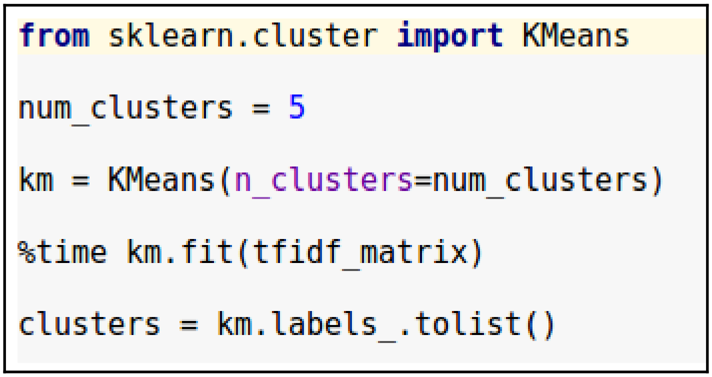


```
Results for SVC(kernel=rbf)
Training time: 6.319218s; Prediction time: 0.680047s
             precision    recall  f1-score   support

        neg       0.86      0.75      0.80       100
        pos       0.78      0.88      0.83       100

avg / total       0.82      0.81      0.81       200

Results for SVC(kernel=linear)
Training time: 5.752379s; Prediction time: 0.565493s
             precision    recall  f1-score   support

        neg       0.91      0.92      0.92       100
        pos       0.92      0.91      0.91       100

avg / total       0.92      0.92      0.91       200

Results for LinearSVC()
Training time: 0.034271s; Prediction time: 0.000185s
             precision    recall  f1-score   support

        neg       0.92      0.94      0.93       100
        pos       0.94      0.92      0.93       100

avg / total       0.93      0.93      0.93       200
```

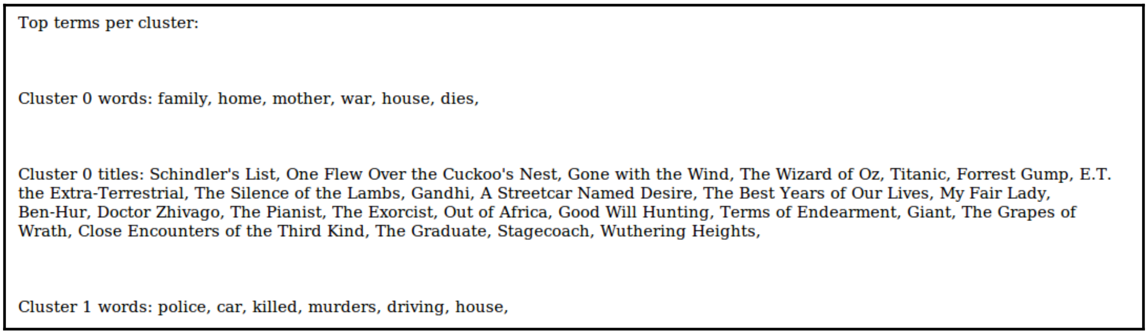




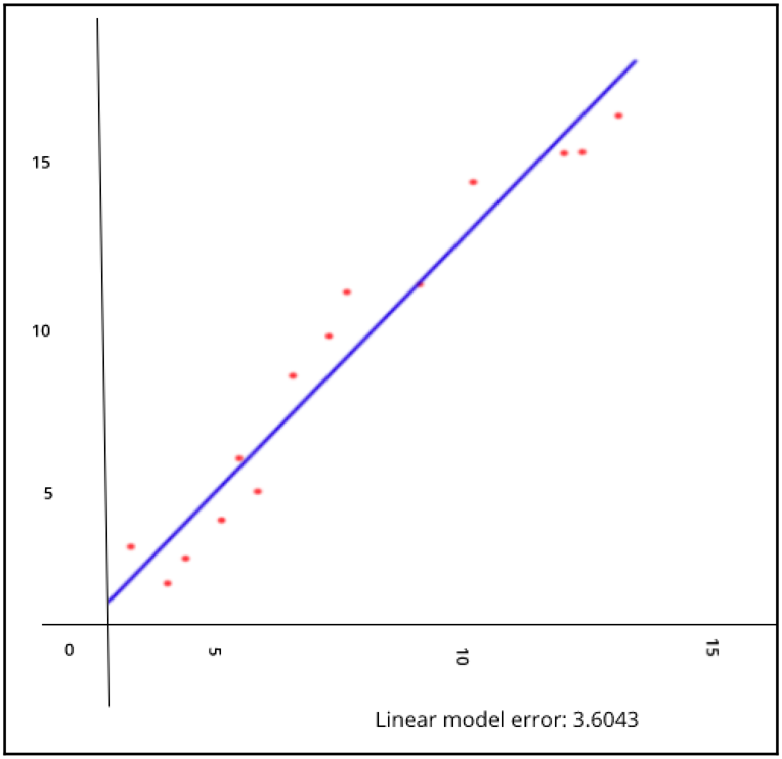
Data Points	X	Y
A	1	1
B	1	0
C	0	2
D	2	4
E	3	5



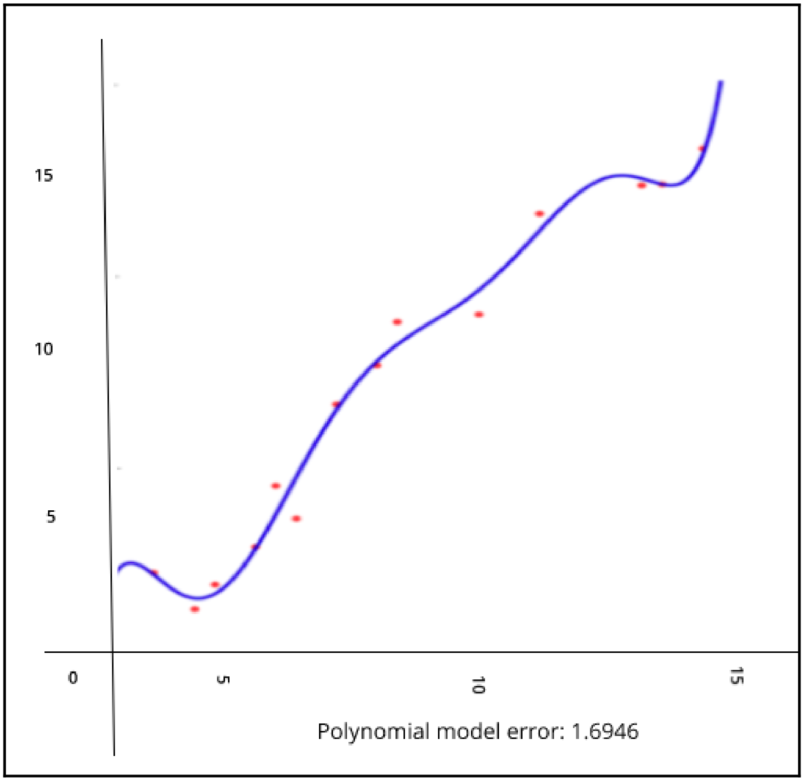
```
from sklearn.cluster import KMeans

num_clusters = 5

km = KMeans(n_clusters=num_clusters)

%time km.fit(tfidf_matrix)

clusters = km.labels_.tolist()
```

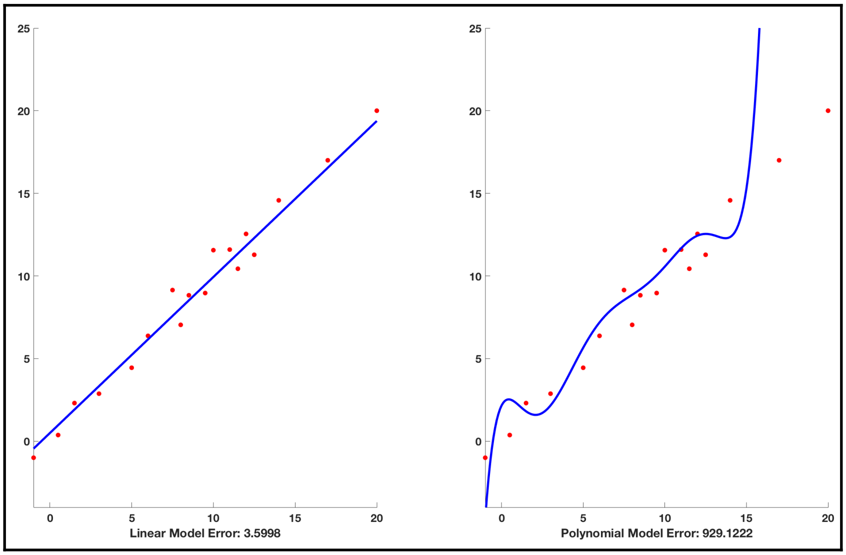


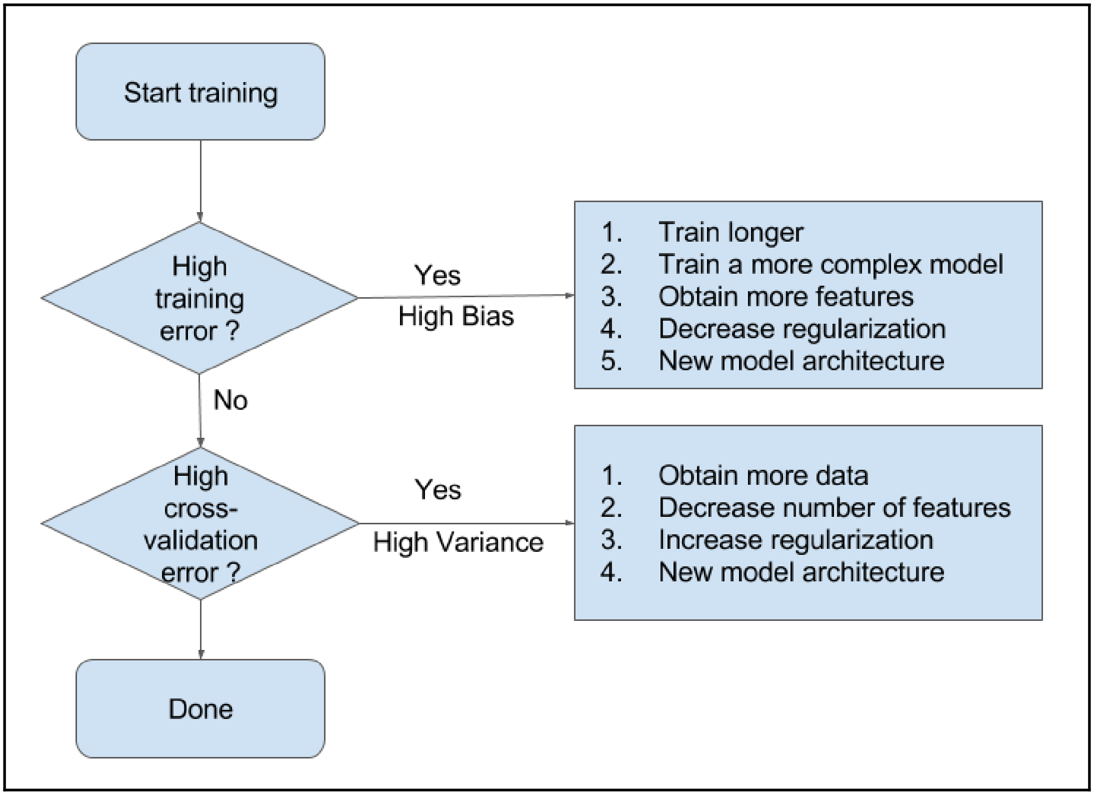
```
Top terms per cluster:


Cluster 0 words: family, home, mother, war, house, dies,


Cluster 0 titles: Schindler's List, One Flew Over the Cuckoo's Nest, Gone with the Wind, The Wizard of Oz, Titanic, Forrest Gump, E.T.
the Extra-Terrestrial, The Silence of the Lambs, Gandhi, A Streetcar Named Desire, The Best Years of Our Lives, My Fair Lady,
Ben-Hur, Doctor Zhivago, The Pianist, The Exorcist, Out of Africa, Good Will Hunting, Terms of Endearment, Giant, The Grapes of
Wrath, Close Encounters of the Third Kind, The Graduate, Stagecoach, Wuthering Heights,


Cluster 1 words: police, car, killed, murders, driving, house,
```

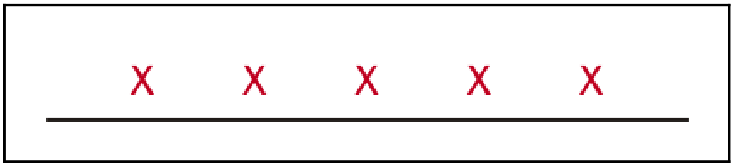


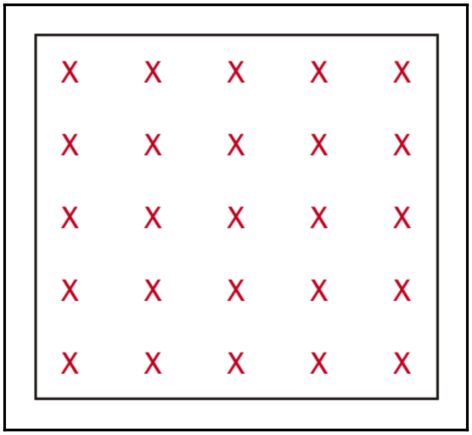
Linear model error: 3.6043

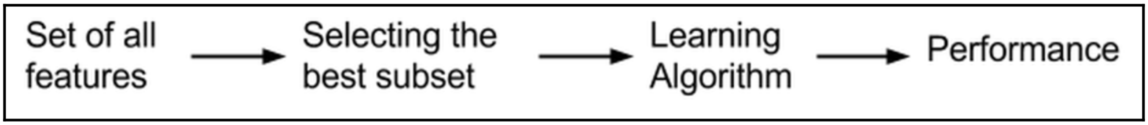

15
10
5
0
5
10
15
Polynomial model error: 1.6946

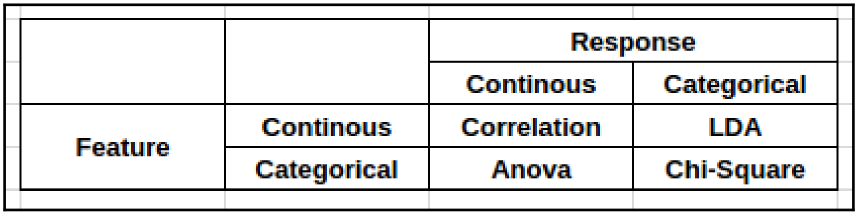

25
20
15
10
5
0
0
5
10
15
20
Linear Model Error: 3.5998
25
20
15
10
5
0
0
5
10
15
20
Polynomial Model Error: 929.1222


Start training
High training error ?
Yes
High Bias
1. Train longer
2. Train a more complex model
3. Obtain more features
4. Decrease regularization
5. New model architecture
No
High cross-validation error ?
Yes
High Variance
1. Obtain more data
2. Decrease number of features
3. Increase regularization
4. New model architecture
Done

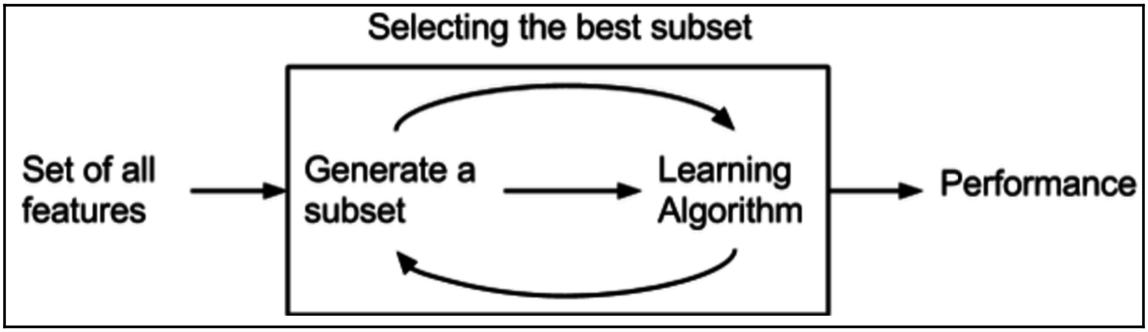

X X X X X

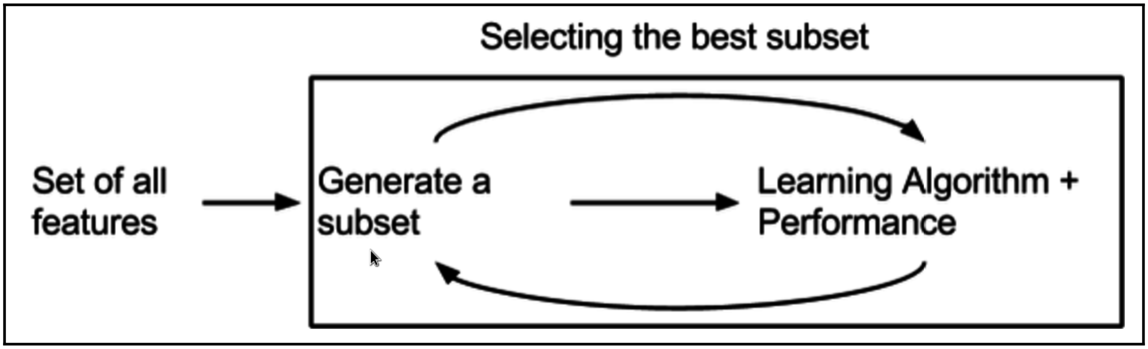






		Response	
		Continous	Categorical
Feature	Continous	Correlation	LDA
	Categorical	Anova	Chi-Square



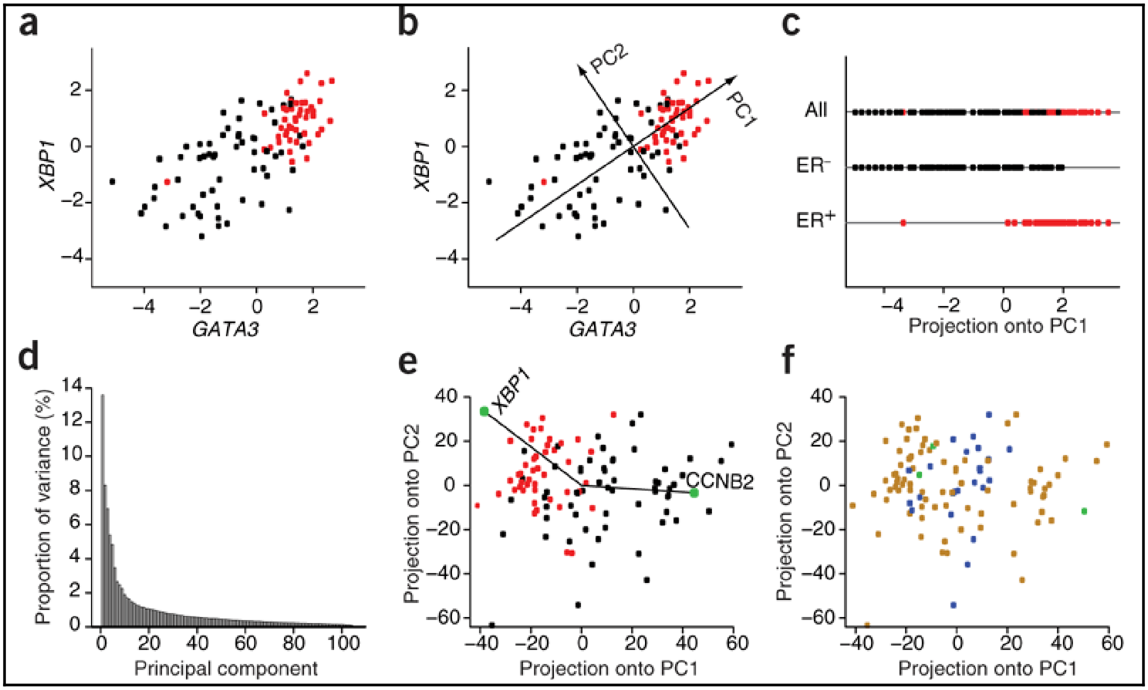


Selecting the best subset

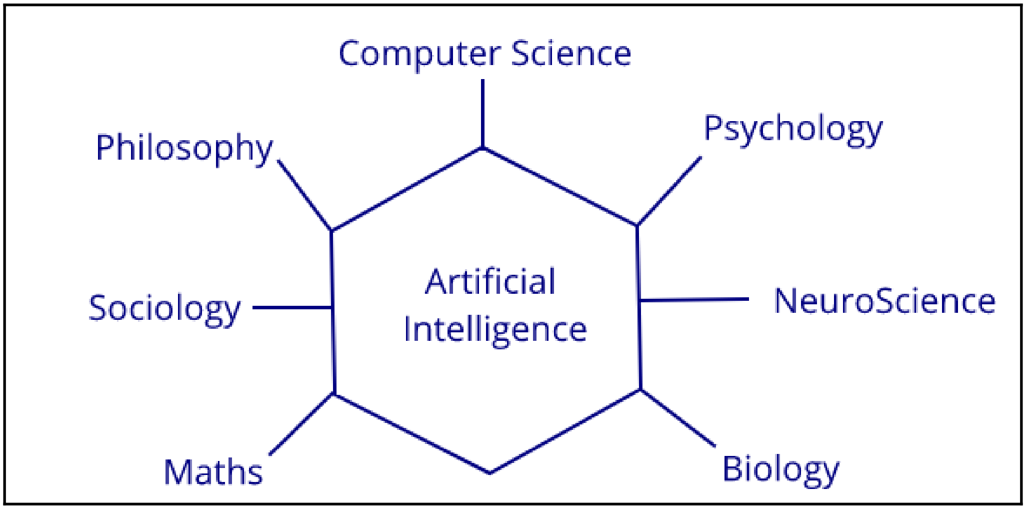
Set of all features

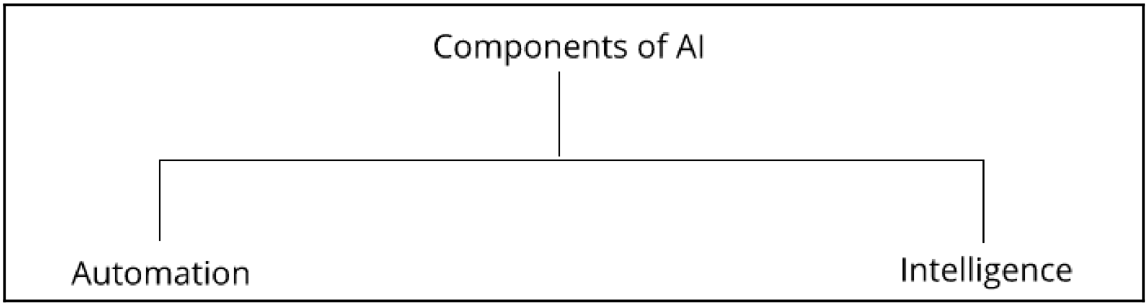
Generate a subset

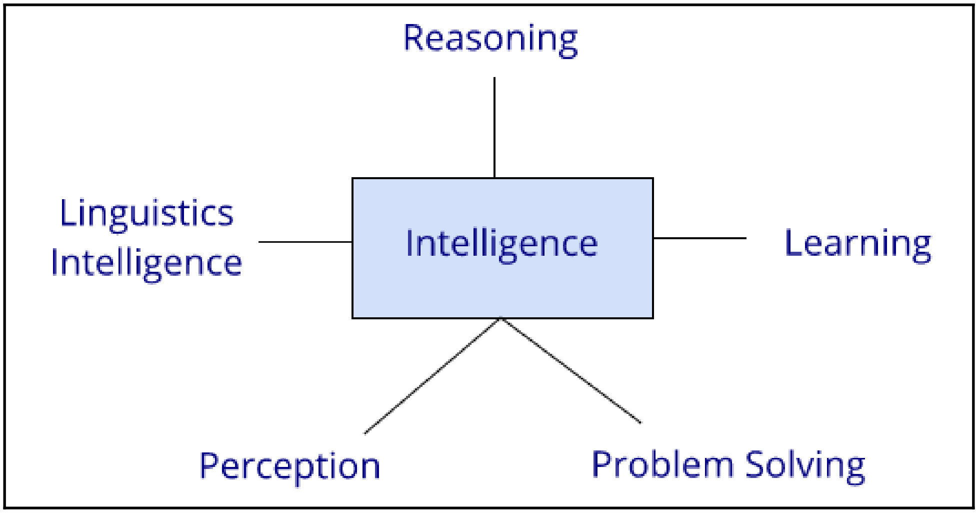
Learning Algorithm + Performance

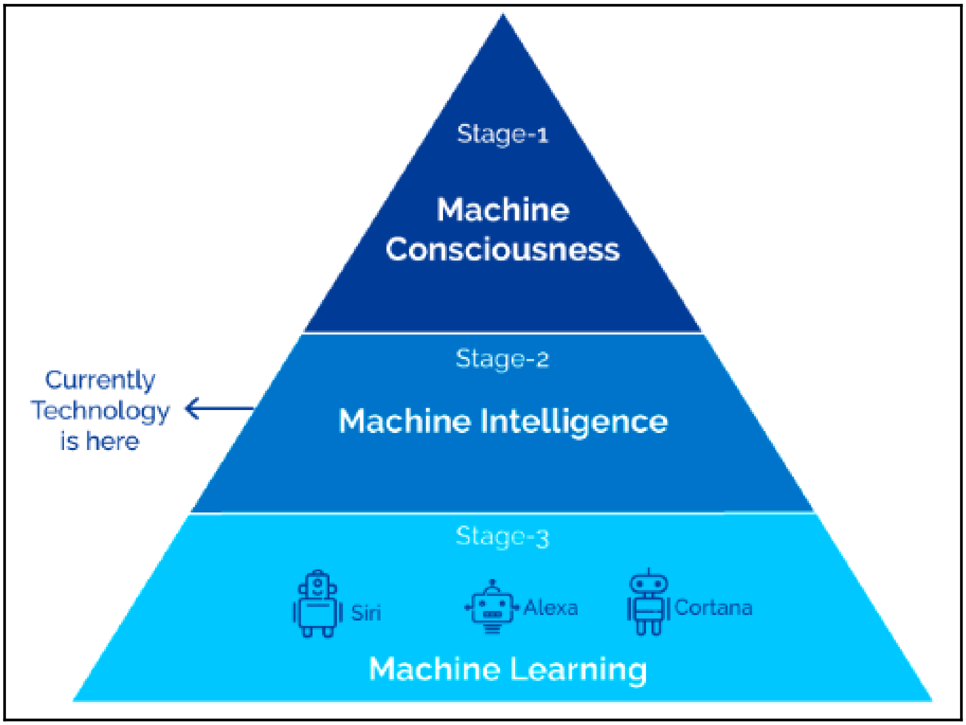


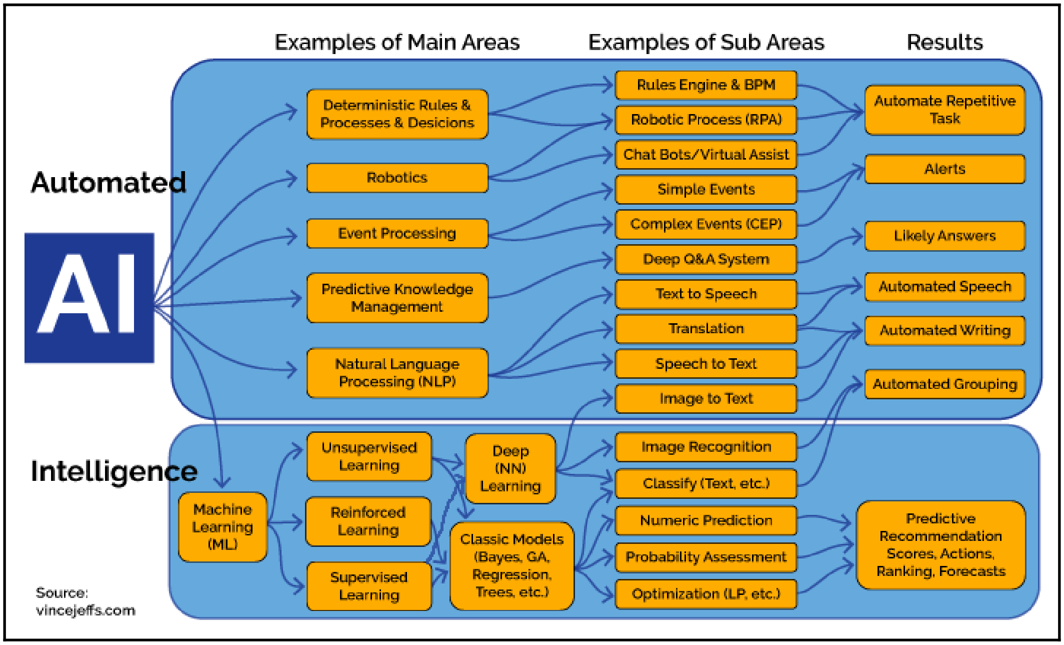
Chapter 9: Deep Learning for NLU and NLG Problems






Reasoning
Linguistics Intelligence
Intelligence
Learning
Perception
Problem Solving


Stage-1
Machine Consciousness
Stage-2
Machine Intelligence
Currently Technology is here
Stage-3
Siri
Alexa
Cortana
Machine Learning



Examples of Main Areas

Examples of Sub Areas

Results

Automated

AI

Intelligence

Deterministic Rules & Processes & Desicions

Robotics

Event Processing

Predictive Knowledge Management

Natural Language Processing (NLP)

Rules Engine & BPM

Robotic Process (RPA)

Chat Bots/Virtual Assist

Simple Events

Complex Events (CEP)

Deep Q&A System

Text to Speech

Translation

Speech to Text

Image to Text

Automate Repetitive Task

Alerts

Likely Answers

Automated Speech

Automated Writing

Automated Grouping

Machine Learning (ML)

Unsupervised Learning

Reinforced Learning

Supervised Learning

Deep (NN) Learning

Classic Models (Bayes, GA, Regression, Trees, etc.)

Image Recognition

Classify (Text, etc.)

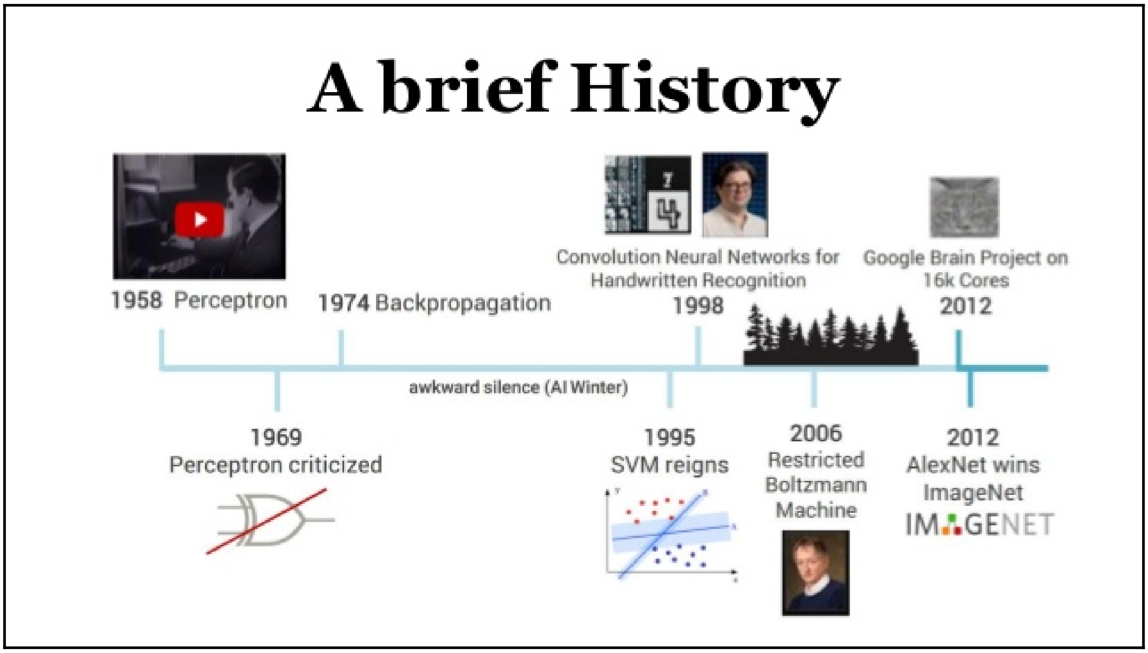
Numeric Prediction

Probability Assessment

Optimization (LP, etc.)

Predictive Recommendation Scores, Actions, Ranking, Forecasts

Source: vincejeffs.com



A brief History

1958 Perceptron

1974 Backpropagation

Convolution Neural Networks for Handwritten Recognition 1998

Google Brain Project on 16k Cores 2012

awkward silence (AI Winter)

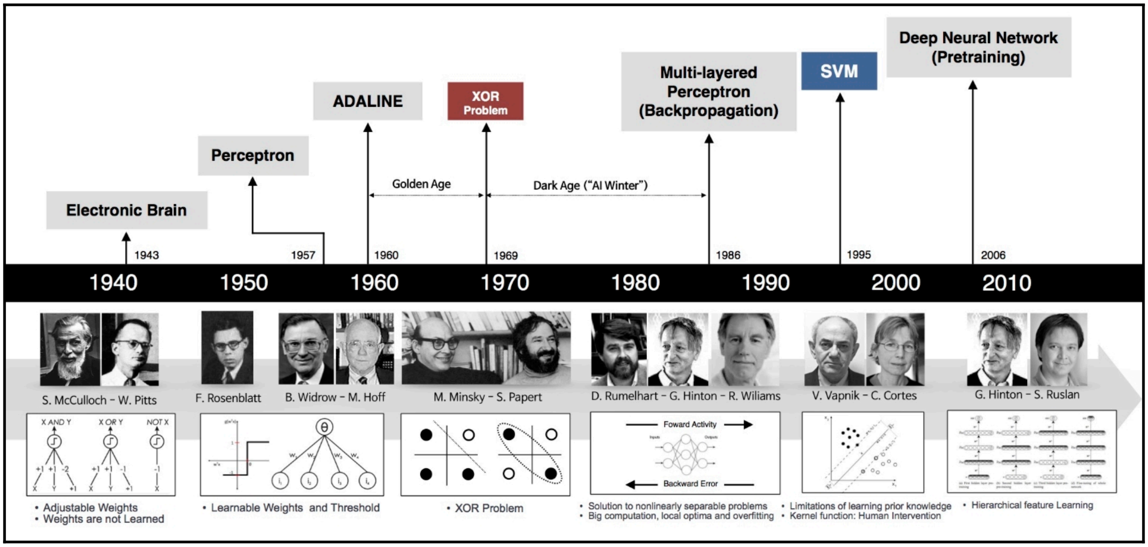
1969 Perceptron criticized

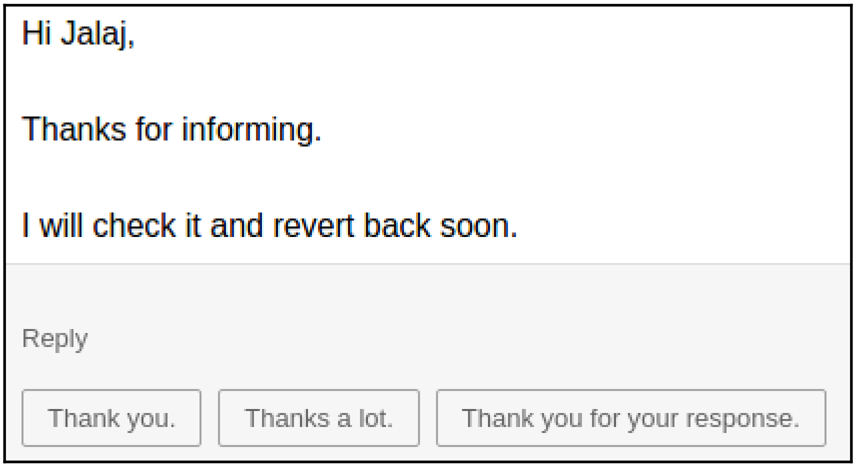
1995 SVM reigns

2006 Restricted Boltzmann Machine

2012 AlexNet wins ImageNet

IMAGENET

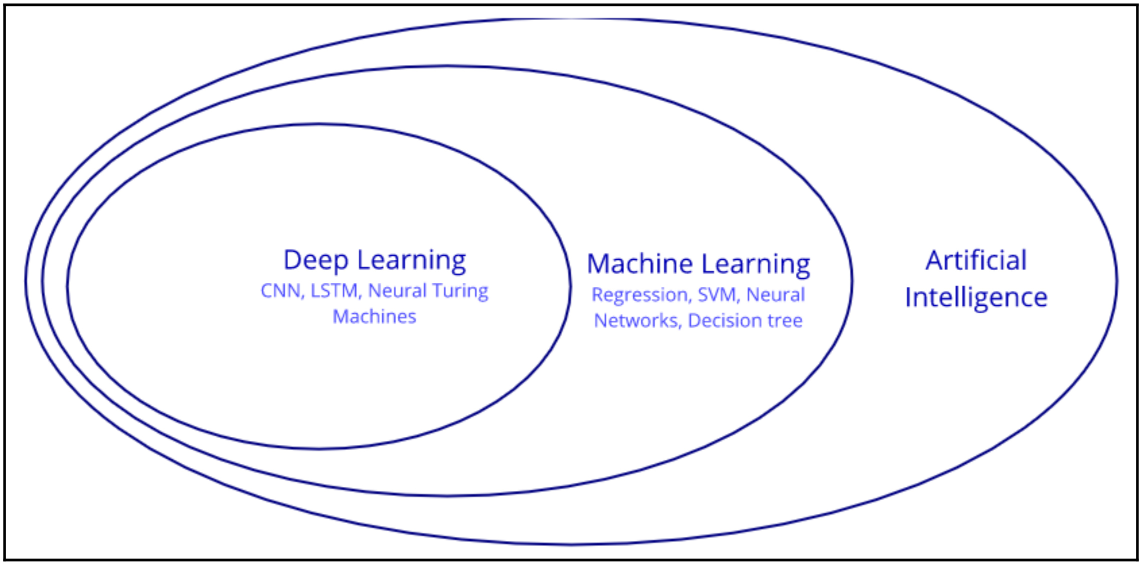

Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

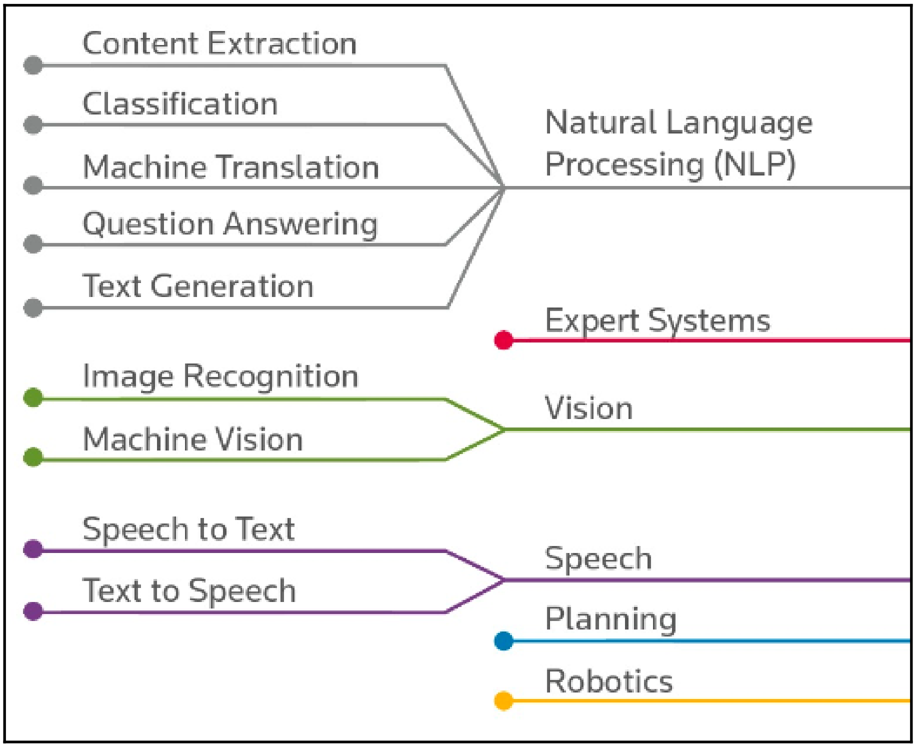


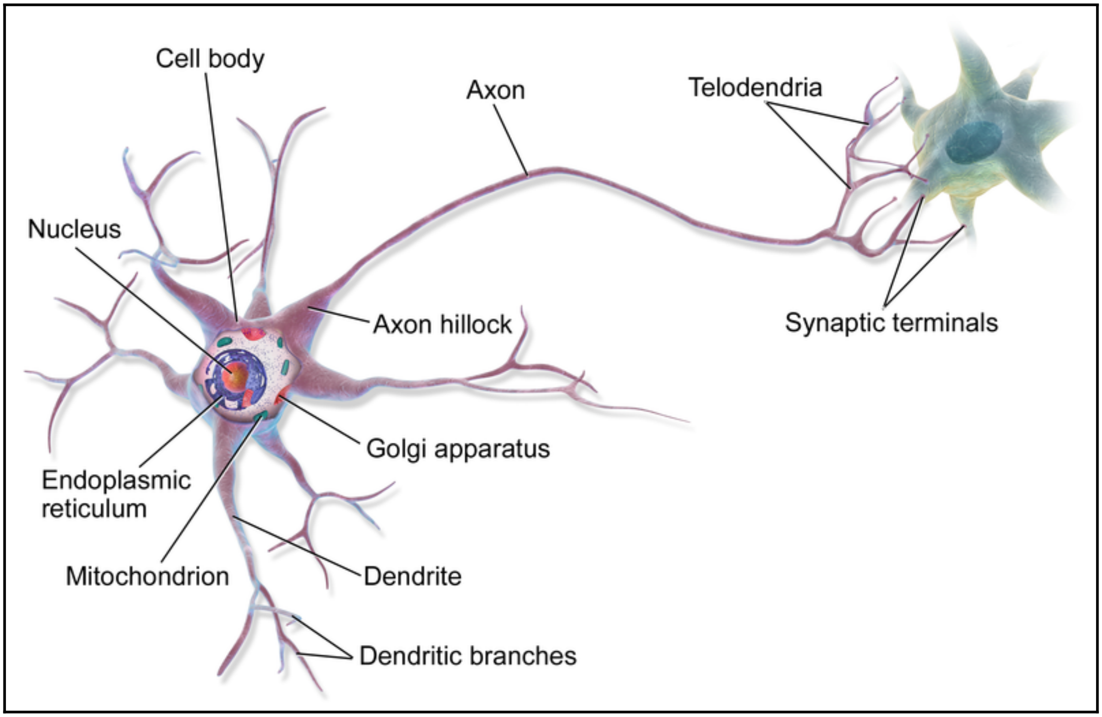
Hi Jalaj,

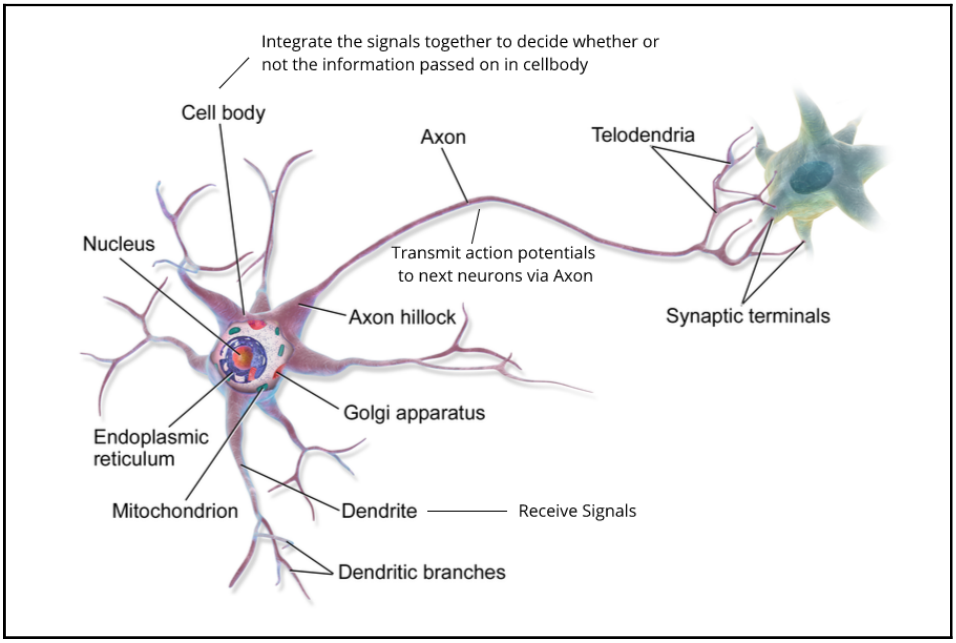
Thanks for informing.

I will check it and revert back soon.


Deep Learning
CNN, LSTM, Neural Turing Machines
Machine Learning
Regression, SVM, Neural Networks, Decision tree
Artificial Intelligence


Content Extraction
Classification
Machine Translation
Question Answering
Text Generation
Natural Language Processing (NLP)
Expert Systems
Image Recognition
Machine Vision
Vision
Speech to Text
Text to Speech
Speech
Planning
Robotics

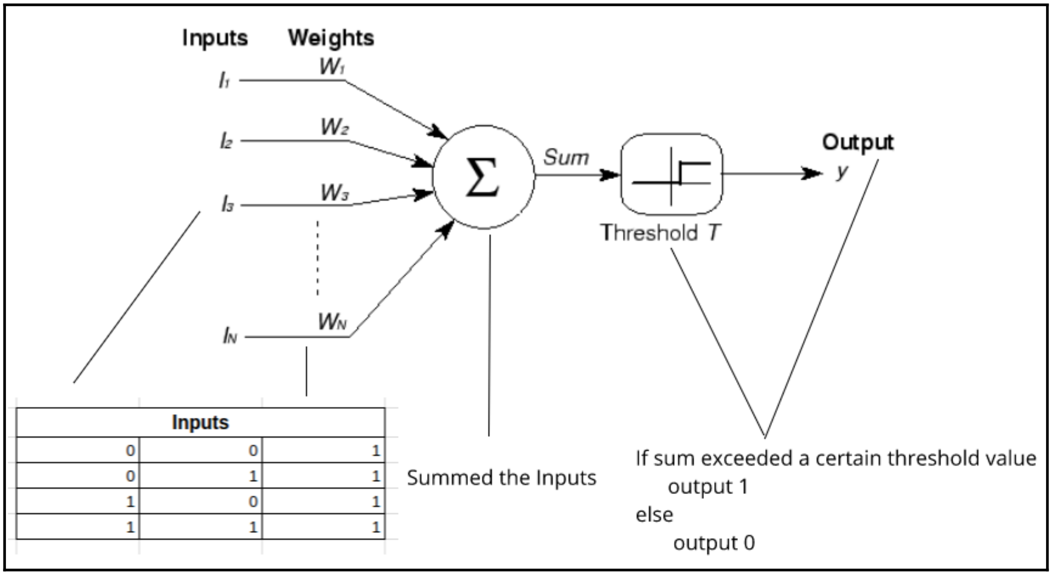




Integrate the signals together to decide whether or not the information passed on in cellbody

Transmit action potentials to next neurons via Axon

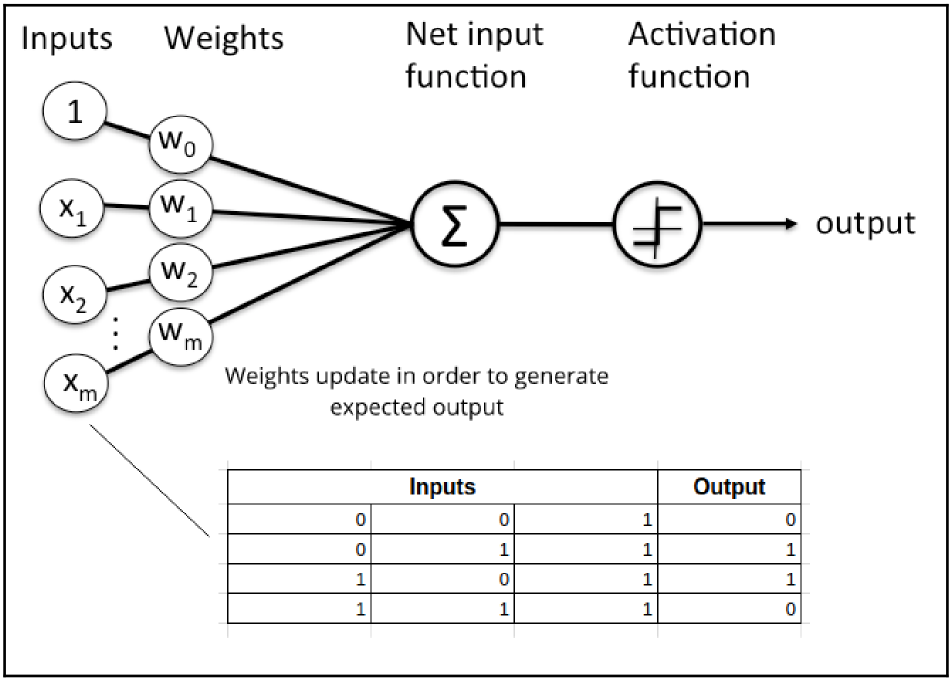
Receive Signals



Inputs		
0	0	1
0	1	1
1	0	1
1	1	1

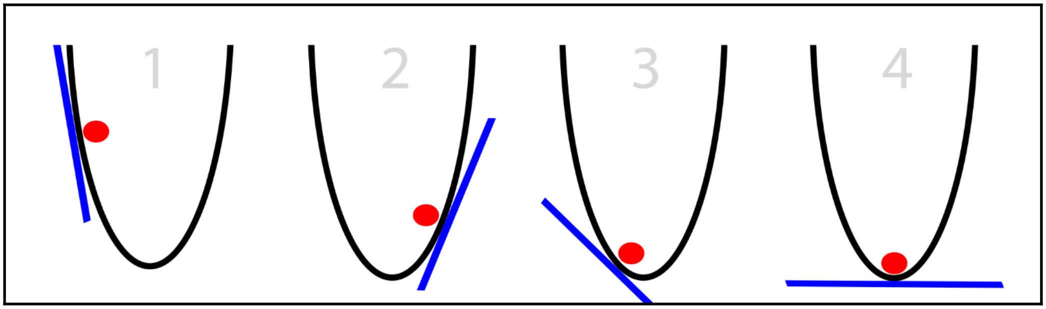
Summed the Inputs

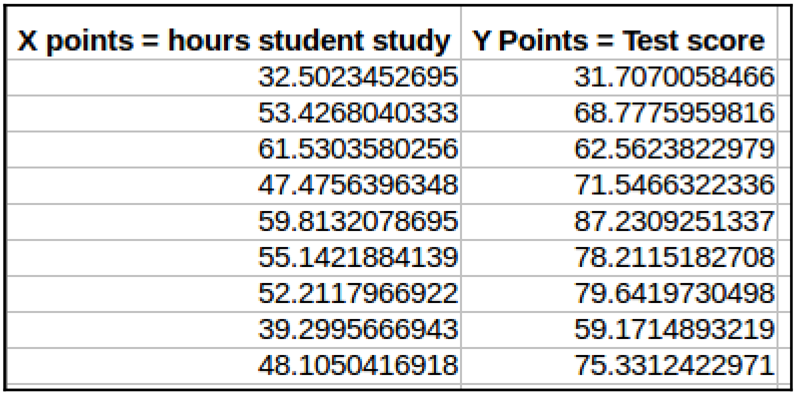
If sum exceeded a certain threshold value
output 1
else
output 0



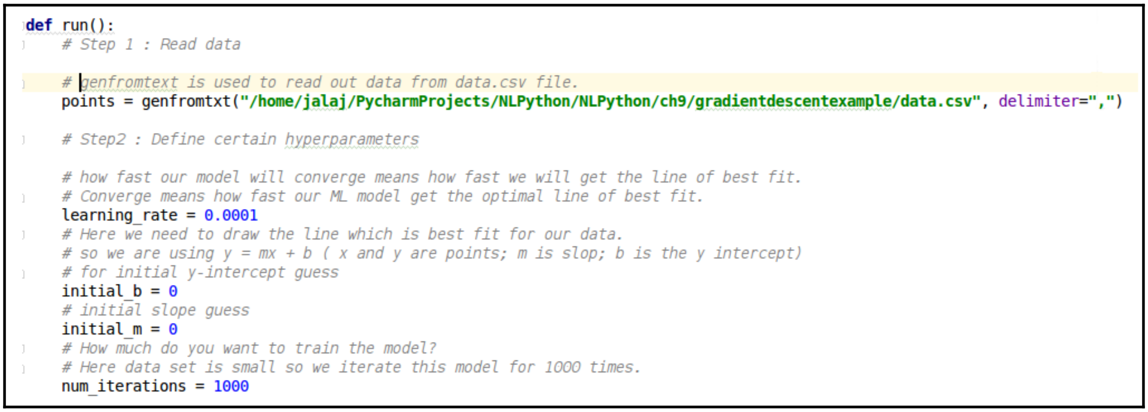
Weights update in order to generate expected output

Inputs			Output
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

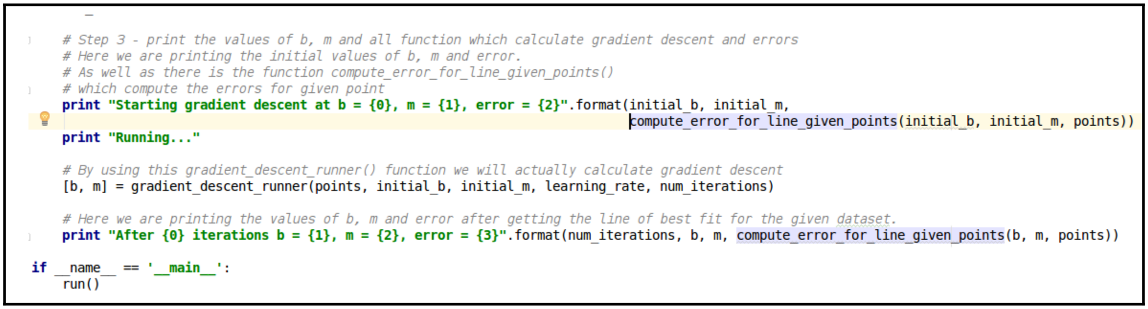




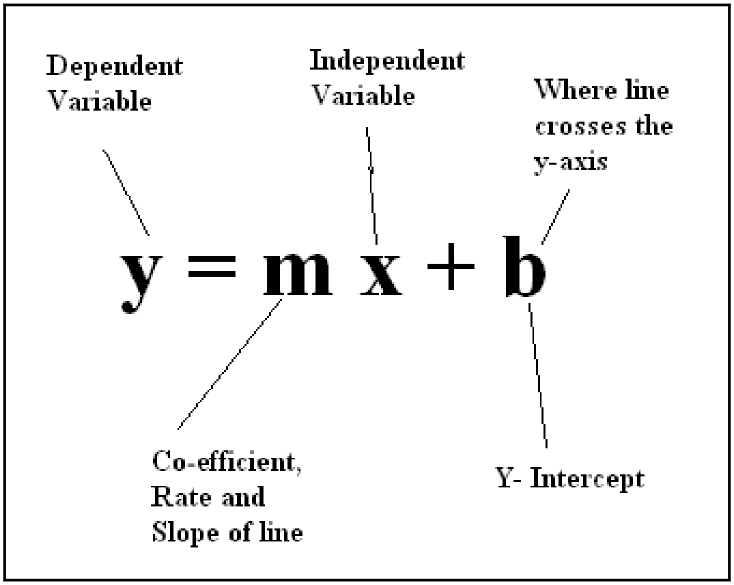
X points = hours student study	Y Points = Test score
32.5023452695	31.7070058466
53.4268040333	68.7775959816
61.5303580256	62.5623822979
47.4756396348	71.5466322336
59.8132078695	87.2309251337
55.1421884139	78.2115182708
52.2117966922	79.6419730498
39.2995666943	59.1714893219
48.1050416918	75.3312422971

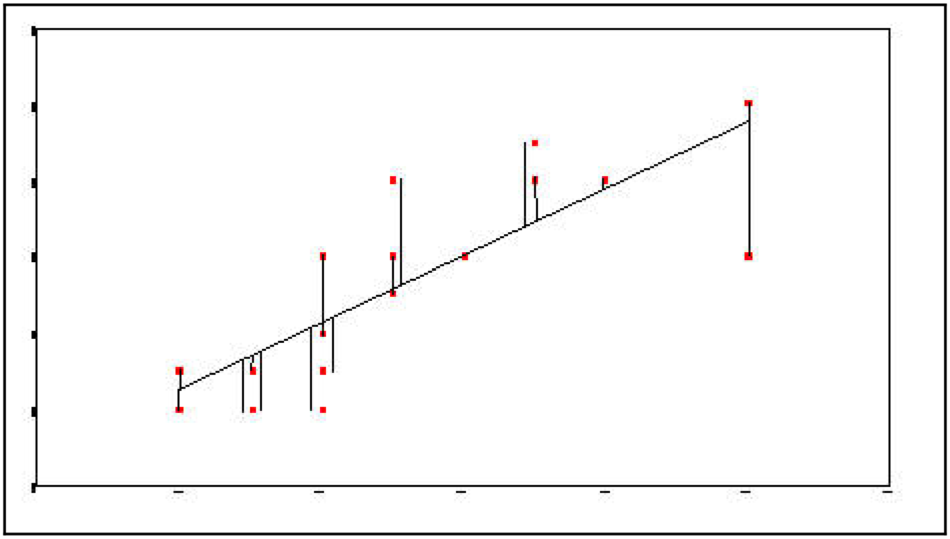


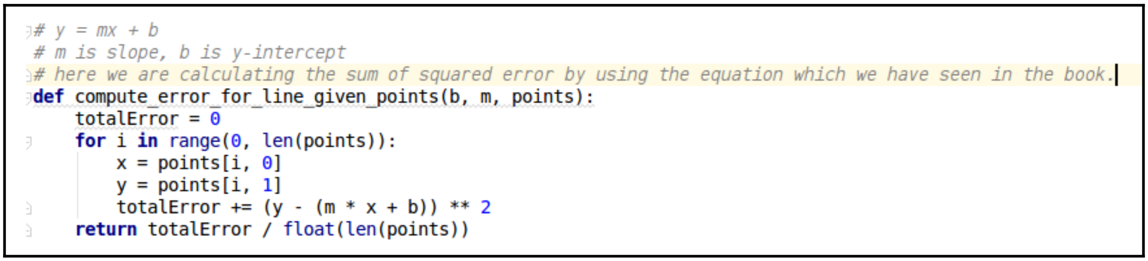
```
def run():
    # Step 1 : Read data

    # genfromtext is used to read out data from data.csv file.
    points = genfromtxt("/home/jalaj/PycharmProjects/NLPython/NLPython/ch9/gradientdescentexample/data.csv", delimiter=",")

    # Step2 : Define certain hyperparameters

    # how fast our model will converge means how fast we will get the line of best fit.
    # Converge means how fast our ML model get the optimal line of best fit.
    learning_rate = 0.0001
    # Here we need to draw the line which is best fit for our data.
    # so we are using y = mx + b ( x and y are points; m is slop; b is the y intercept)
    # for initial y-intercept guess
    initial_b = 0
    # initial slope guess
    initial_m = 0
    # How much do you want to train the model?
    # Here data set is small so we iterate this model for 1000 times.
    num_iterations = 1000
```



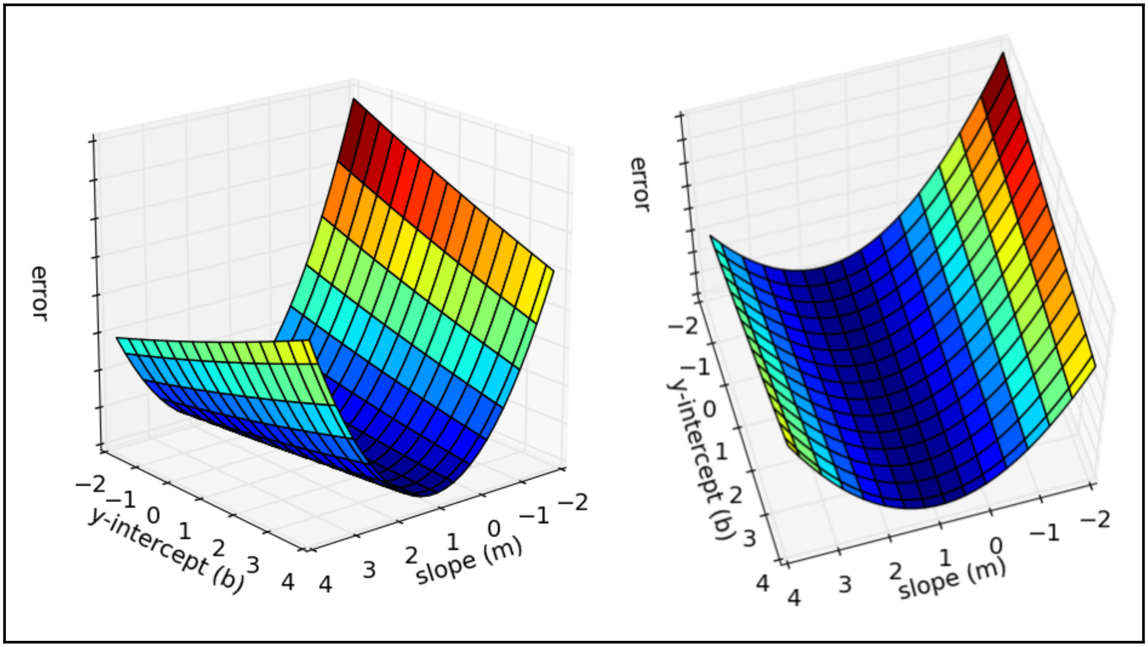
```
    # Step 3 - print the values of b, m and all function which calculate gradient descent and errors
    # Here we are printing the initial values of b, m and error.
    # As well as there is the function compute_error_for_line_given_points()
    # which compute the errors for given point
    print "Starting gradient descent at b = {0}, m = {1}, error = {2}".format(initial_b, initial_m,
                                                                              compute_error_for_line_given_points(initial_b, initial_m, points))
    print "Running..."

    # By using this gradient_descent_runner() function we will actually calculate gradient descent
    [b, m] = gradient_descent_runner(points, initial_b, initial_m, learning_rate, num_iterations)

    # Here we are printing the values of b, m and error after getting the line of best fit for the given dataset.
    print "After {0} iterations b = {1}, m = {2}, error = {3}".format(num_iterations, b, m, compute_error_for_line_given_points(b, m, points))

if __name__ == '__main__':
    run()
```

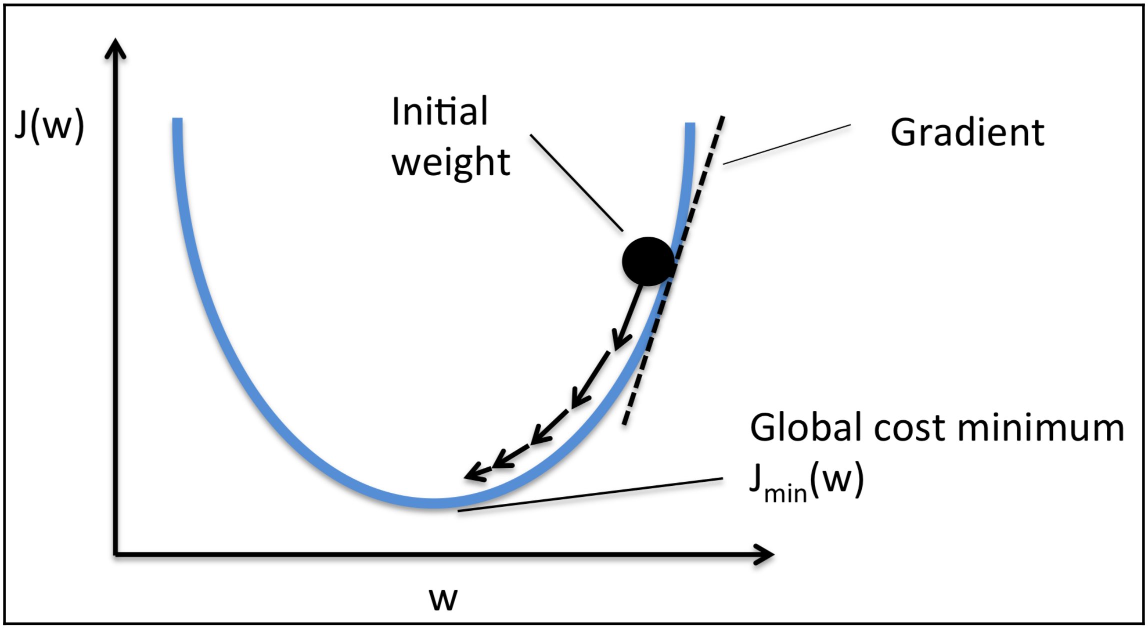

Dependent Variable
Independent Variable
Where line crosses the y-axis
y = m x + b
Co-efficient, Rate and Slope of line
Y- Intercept

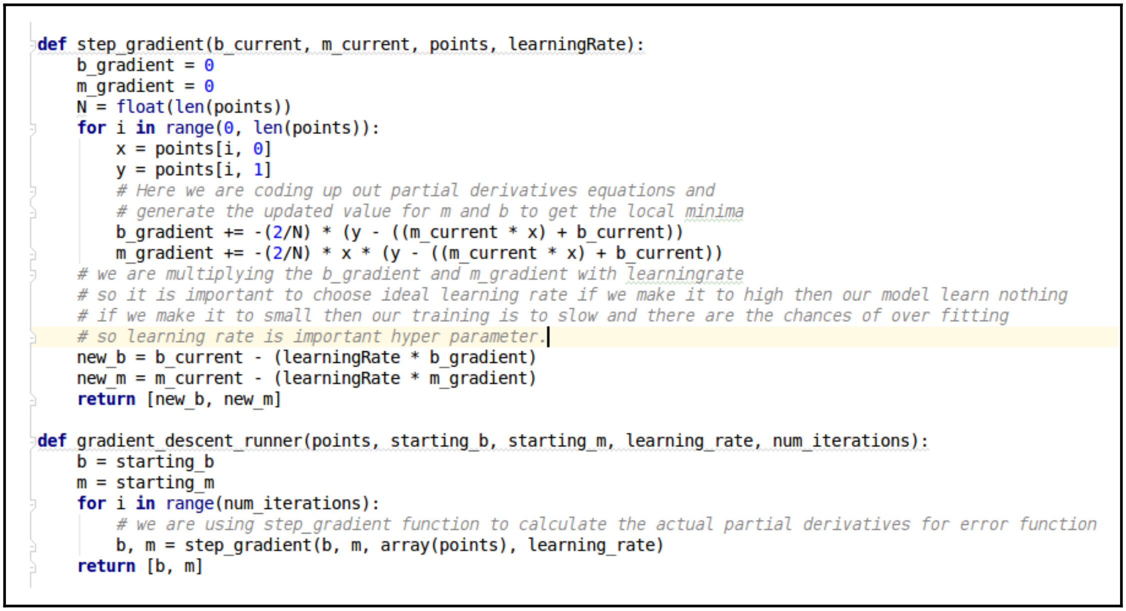




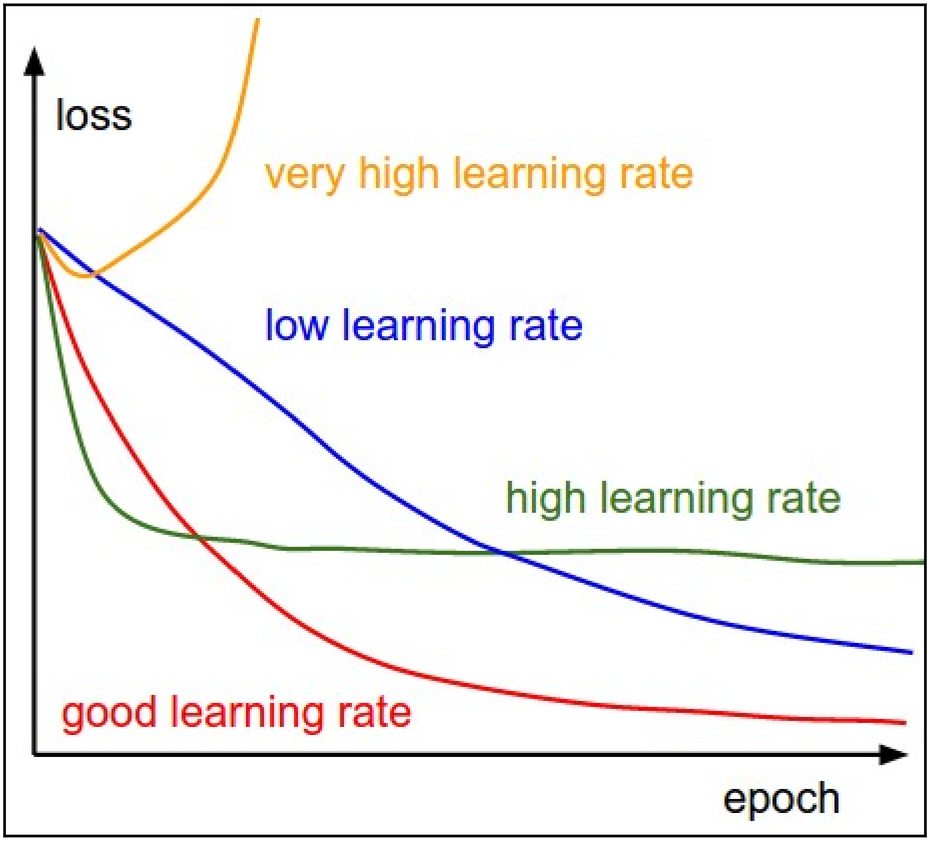
```
# y = mx + b
# m is slope, b is y-intercept
# here we are calculating the sum of squared error by using the equation which we have seen in the book.
def compute_error_for_line_given_points(b, m, points):
    totalError = 0
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        totalError += (y - (m * x + b)) ** 2
    return totalError / float(len(points))
```

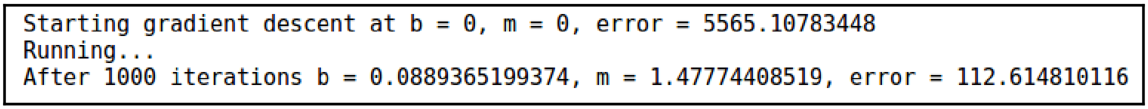

error
y-intercept (b)
slope (m)
−2 −1 0 1 2 3 4
4 3 2 1 0 −1 −2
error
y-intercept (b)
slope (m)
−2 −1 0 1 2 3 4
4 3 2 1 0 −1 −2


J(w)
Initial weight
Gradient
Global cost minimum
J$_{min}$(w)
w

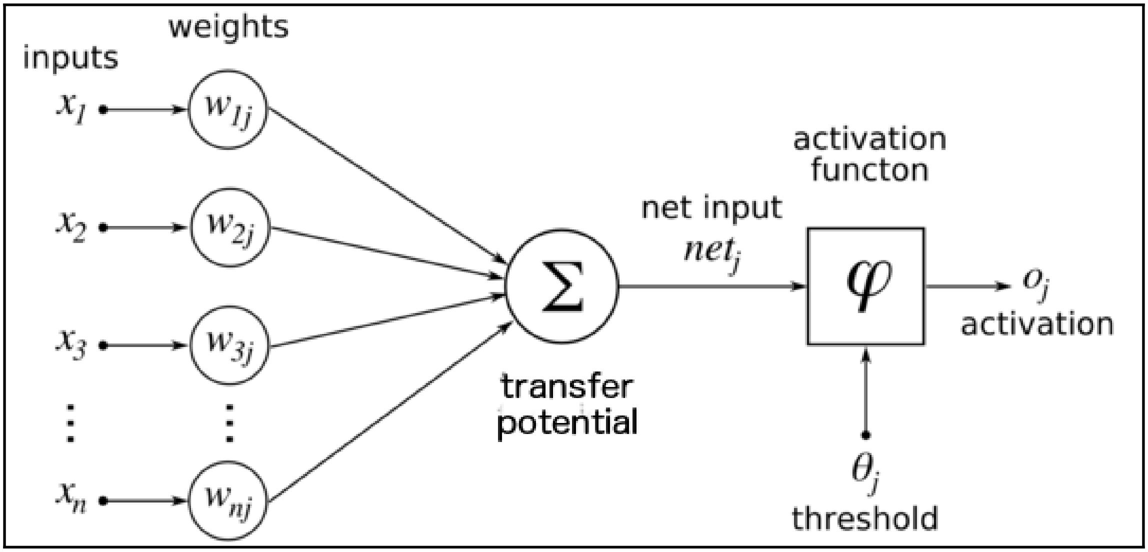


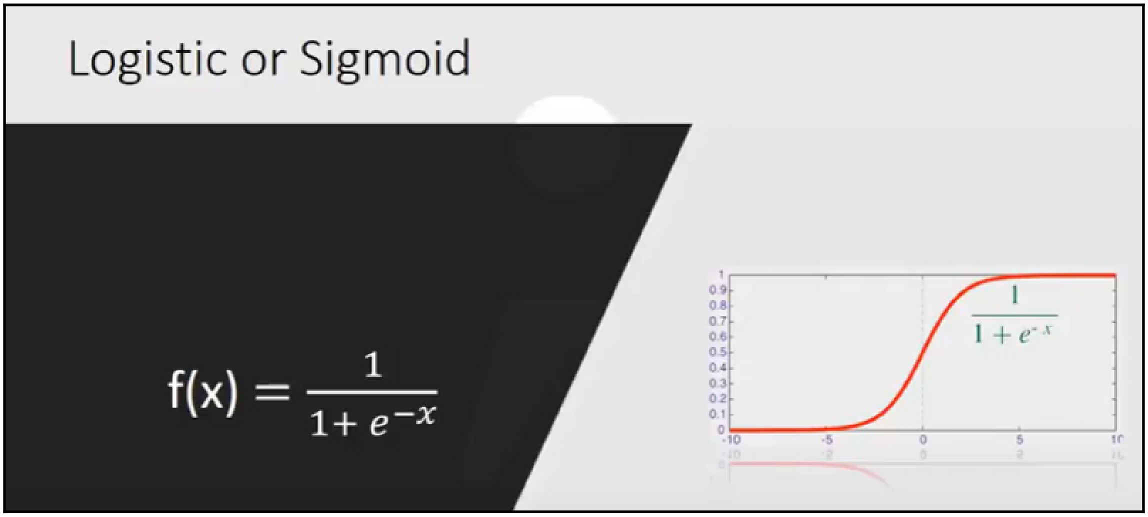
```
def step_gradient(b_current, m_current, points, learningRate):
    b_gradient = 0
    m_gradient = 0
    N = float(len(points))
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        # Here we are coding up out partial derivatives equations and
        # generate the updated value for m and b to get the local minima
        b_gradient += -(2/N) * (y - ((m_current * x) + b_current))
        m_gradient += -(2/N) * x * (y - ((m_current * x) + b_current))
    # we are multiplying the b_gradient and m_gradient with learningrate
    # so it is important to choose ideal learning rate if we make it to high then our model learn nothing
    # if we make it to small then our training is to slow and there are the chances of over fitting
    # so learning rate is important hyper parameter.
    new_b = b_current - (learningRate * b_gradient)
    new_m = m_current - (learningRate * m_gradient)
    return [new_b, new_m]

def gradient_descent_runner(points, starting_b, starting_m, learning_rate, num_iterations):
    b = starting_b
    m = starting_m
    for i in range(num_iterations):
        # we are using step_gradient function to calculate the actual partial derivatives for error function
        b, m = step_gradient(b, m, array(points), learning_rate)
    return [b, m]
```





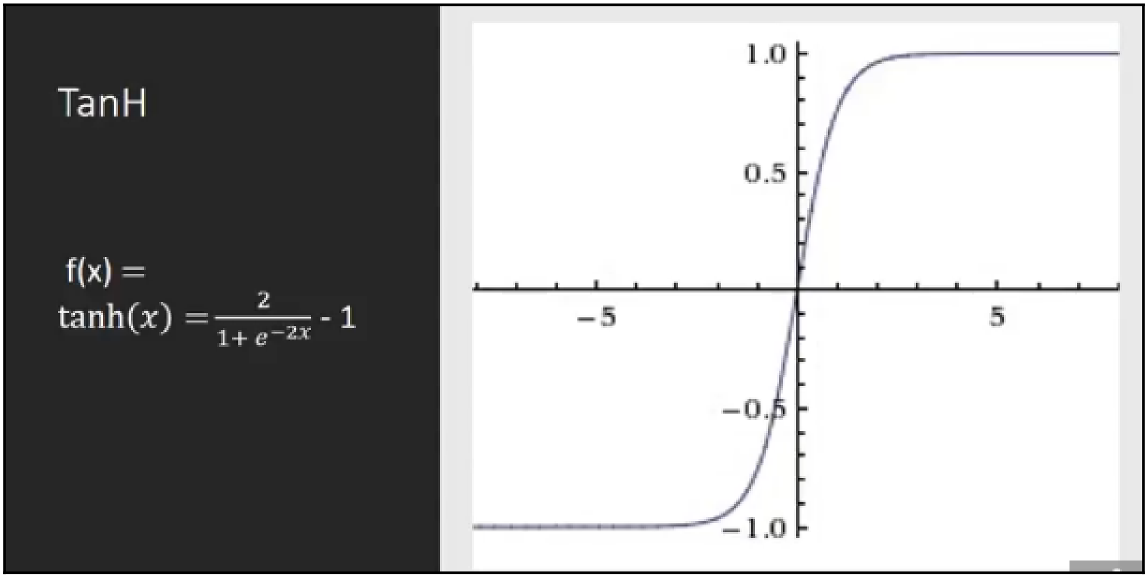
```
Starting gradient descent at b = 0, m = 0, error = 5565.10783448
Running...
After 1000 iterations b = 0.0889365199374, m = 1.47774408519, error = 112.614810116
```





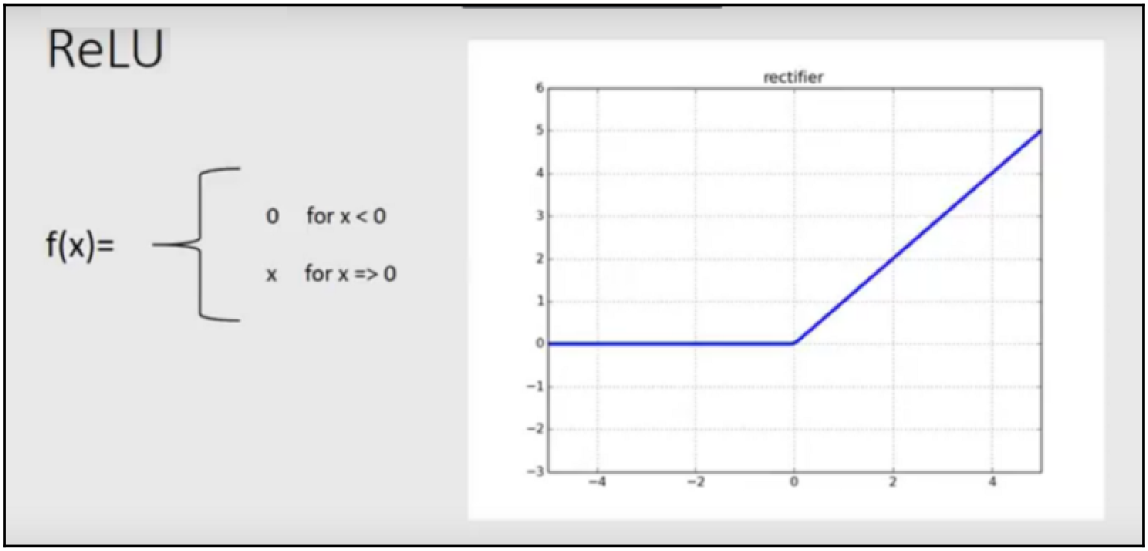
Logistic or Sigmoid

$$f(x) = \frac{1}{1+ e^{-x}}$$



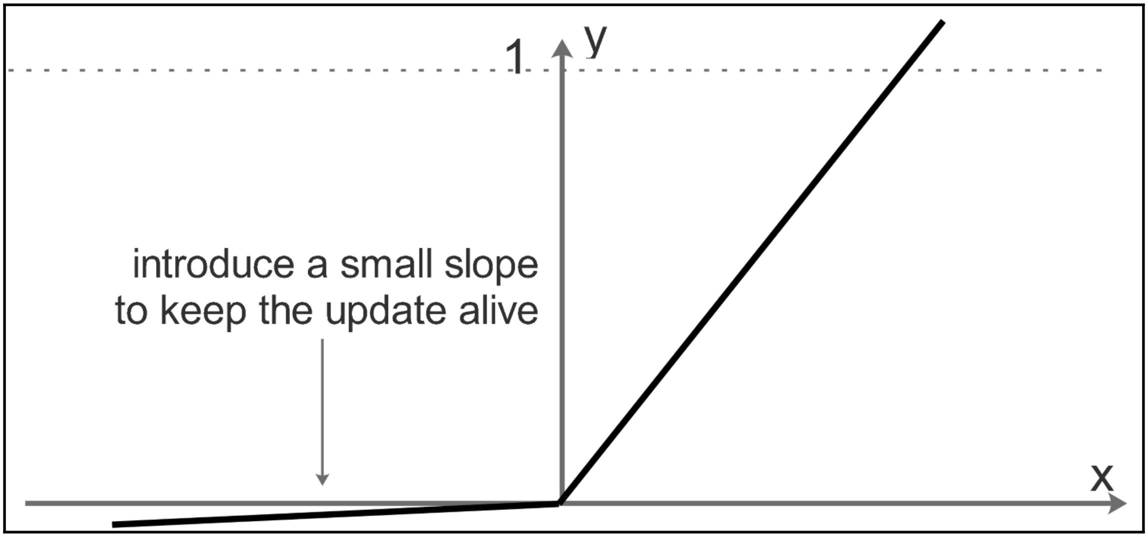
TanH

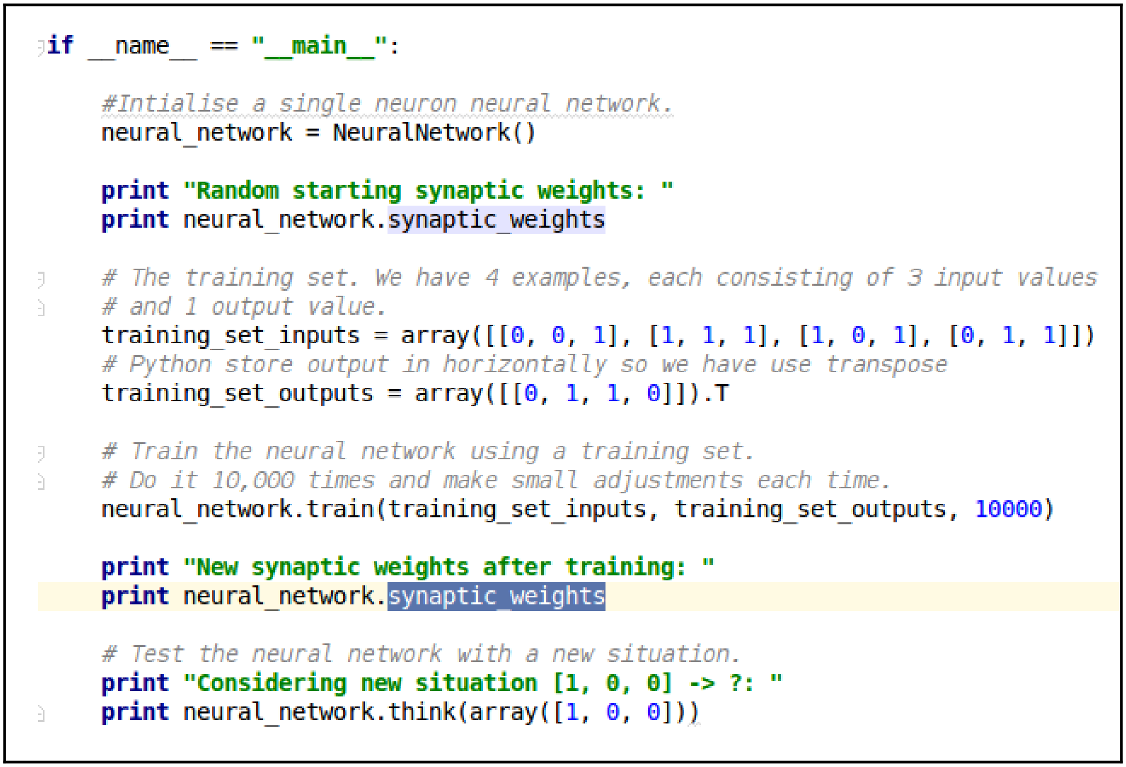
$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



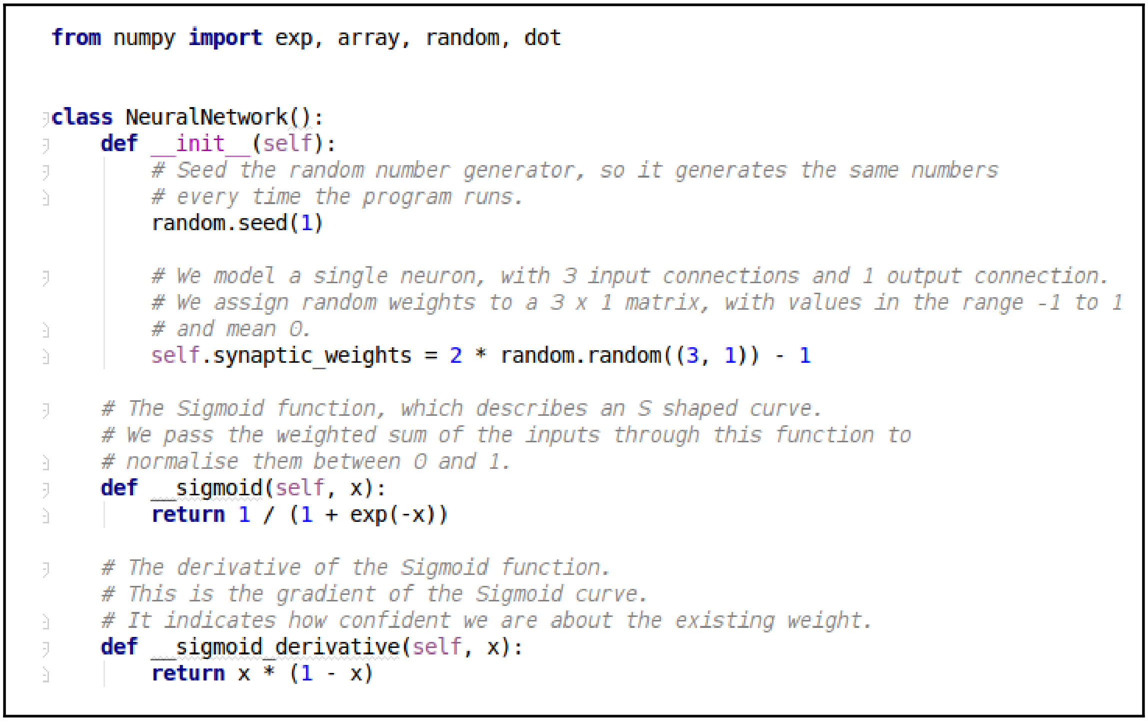
ReLU

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x => 0 \end{cases}$$

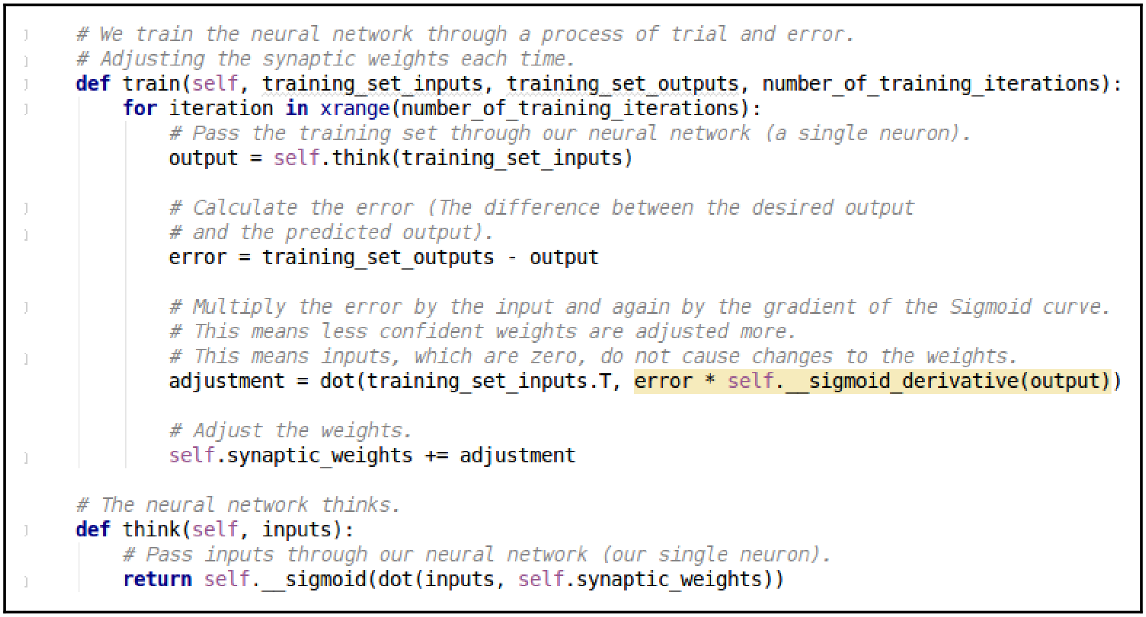




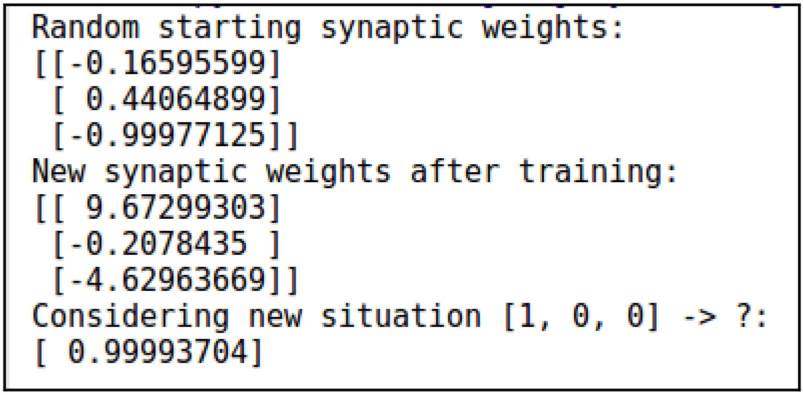
```
if __name__ == "__main__":

    #Intialise a single neuron neural network.
    neural_network = NeuralNetwork()

    print "Random starting synaptic weights: "
    print neural_network.synaptic_weights

    # The training set. We have 4 examples, each consisting of 3 input values
    # and 1 output value.
    training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    # Python store output in horizontally so we have use transpose
    training_set_outputs = array([[0, 1, 1, 0]]).T

    # Train the neural network using a training set.
    # Do it 10,000 times and make small adjustments each time.
    neural_network.train(training_set_inputs, training_set_outputs, 10000)

    print "New synaptic weights after training: "
    print neural_network.synaptic_weights

    # Test the neural network with a new situation.
    print "Considering new situation [1, 0, 0] -> ?: "
    print neural_network.think(array([1, 0, 0]))
```



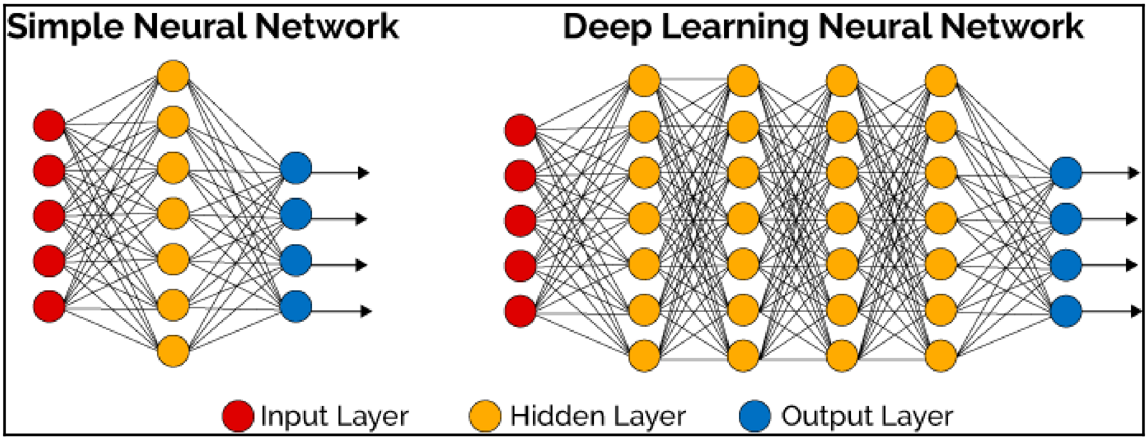
```python
from numpy import exp, array, random, dot


class NeuralNetwork():
    def __init__(self):
        # Seed the random number generator, so it generates the same numbers
        # every time the program runs.
        random.seed(1)

        # We model a single neuron, with 3 input connections and 1 output connection.
        # We assign random weights to a 3 x 1 matrix, with values in the range -1 to 1
        # and mean 0.
        self.synaptic_weights = 2 * random.random((3, 1)) - 1

    # The Sigmoid function, which describes an S shaped curve.
    # We pass the weighted sum of the inputs through this function to
    # normalise them between 0 and 1.
    def __sigmoid(self, x):
        return 1 / (1 + exp(-x))

    # The derivative of the Sigmoid function.
    # This is the gradient of the Sigmoid curve.
    # It indicates how confident we are about the existing weight.
    def __sigmoid_derivative(self, x):
        return x * (1 - x)
```

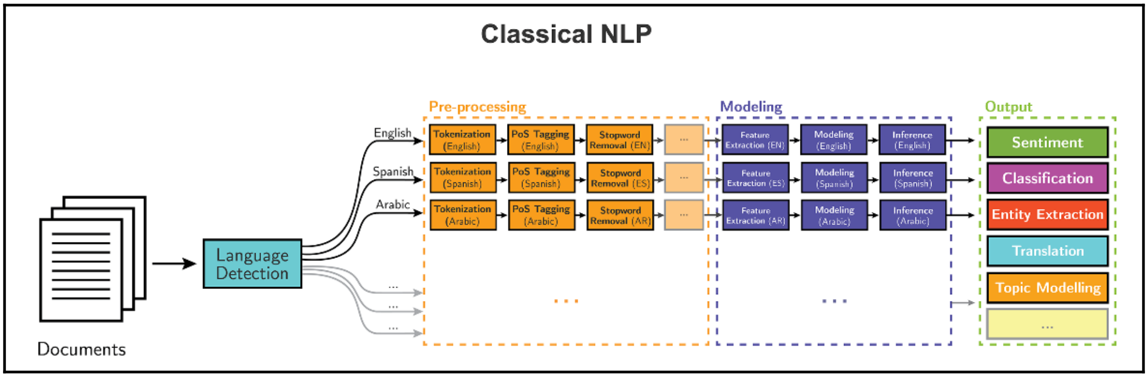


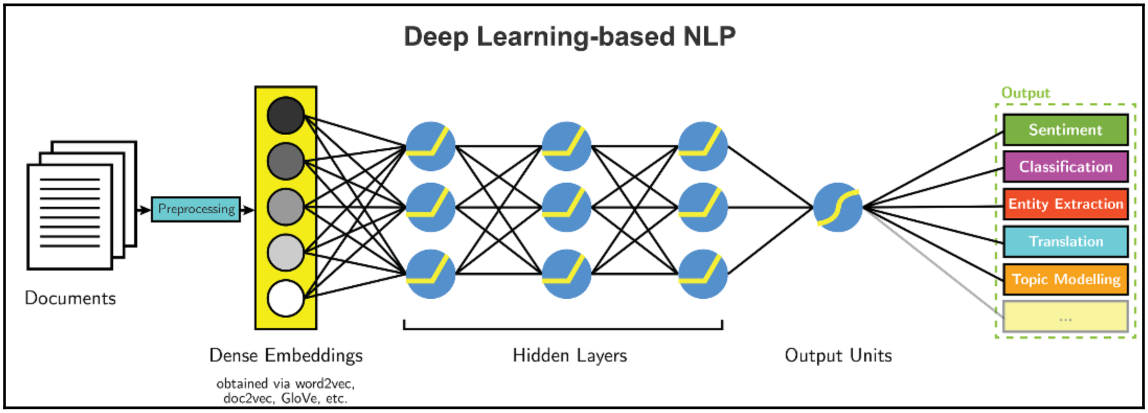
```python
    # We train the neural network through a process of trial and error.
    # Adjusting the synaptic weights each time.
    def train(self, training_set_inputs, training_set_outputs, number_of_training_iterations):
        for iteration in xrange(number_of_training_iterations):
            # Pass the training set through our neural network (a single neuron).
            output = self.think(training_set_inputs)

            # Calculate the error (The difference between the desired output
            # and the predicted output).
            error = training_set_outputs - output

            # Multiply the error by the input and again by the gradient of the Sigmoid curve.
            # This means less confident weights are adjusted more.
            # This means inputs, which are zero, do not cause changes to the weights.
            adjustment = dot(training_set_inputs.T, error * self.__sigmoid_derivative(output))

            # Adjust the weights.
            self.synaptic_weights += adjustment

    # The neural network thinks.
    def think(self, inputs):
        # Pass inputs through our neural network (our single neuron).
        return self.__sigmoid(dot(inputs, self.synaptic_weights))
```

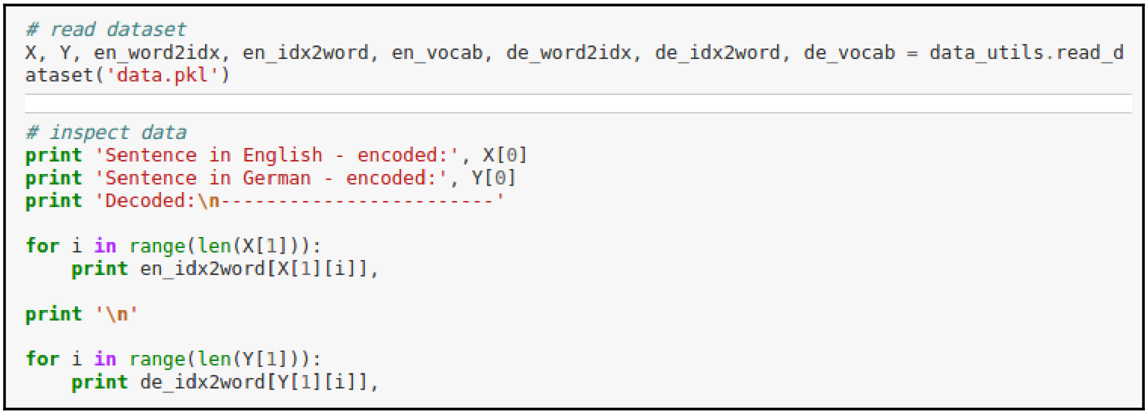


```
Random starting synaptic weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
New synaptic weights after training:
[[ 9.67299303]
 [-0.2078435 ]
 [-4.62963669]]
Considering new situation [1, 0, 0] -> ?:
[ 0.99993704]
```

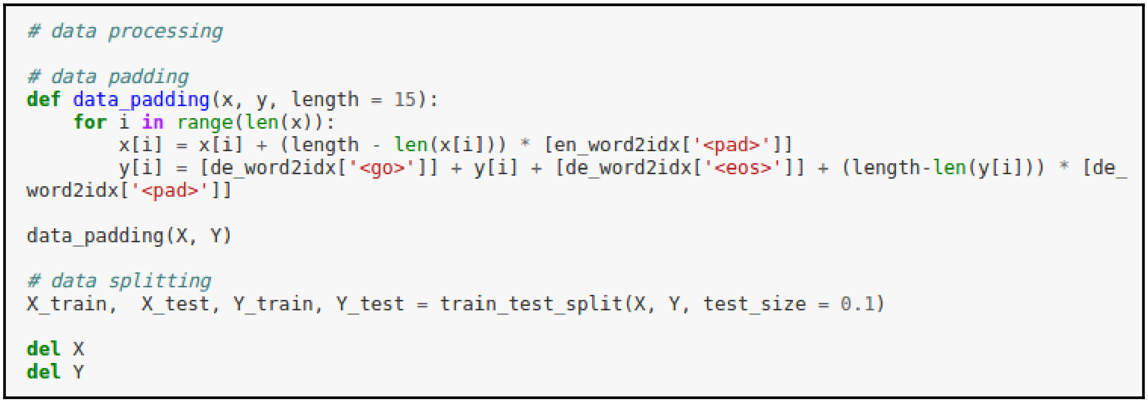




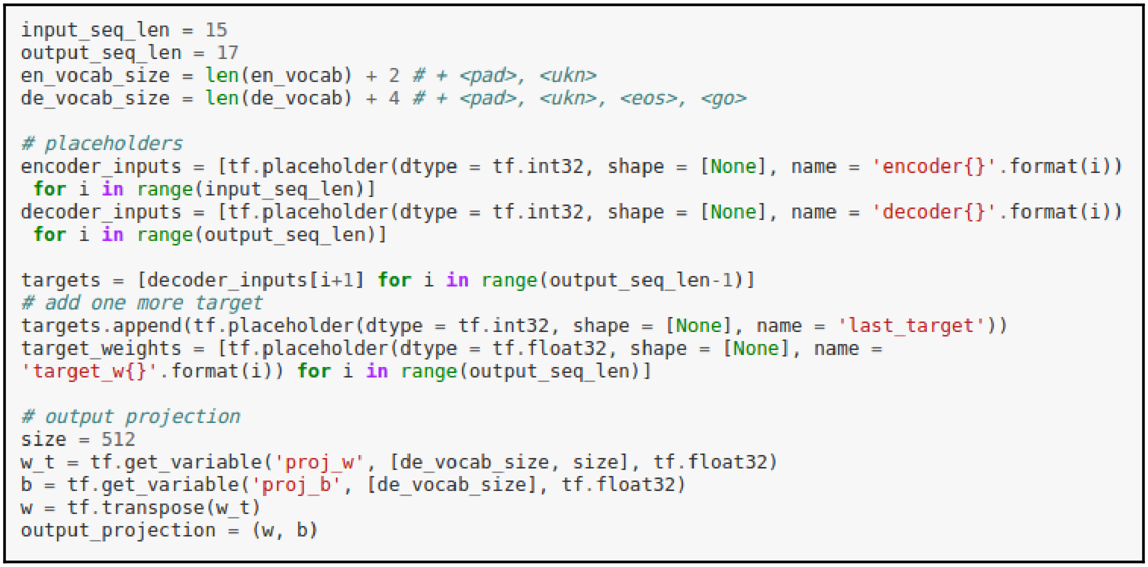




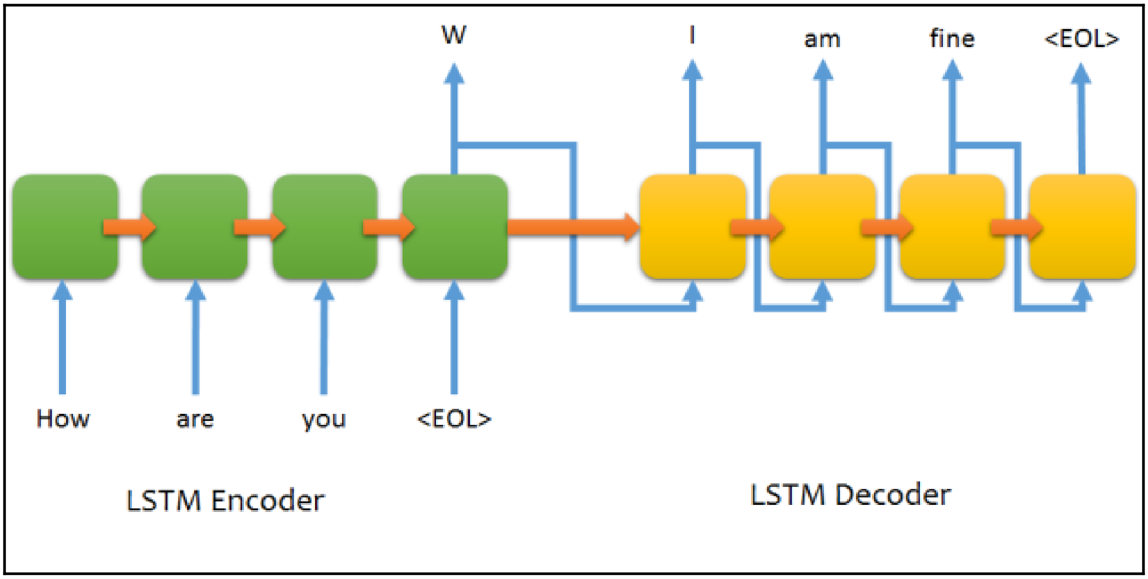
```
# read dataset
X, Y, en_word2idx, en_idx2word, en_vocab, de_word2idx, de_idx2word, de_vocab = data_utils.read_d
ataset('data.pkl')

# inspect data
print 'Sentence in English - encoded:', X[0]
print 'Sentence in German - encoded:', Y[0]
print 'Decoded:\n------------------------'

for i in range(len(X[1])):
    print en_idx2word[X[1][i]],

print '\n'

for i in range(len(Y[1])):
    print de_idx2word[Y[1][i]],
```

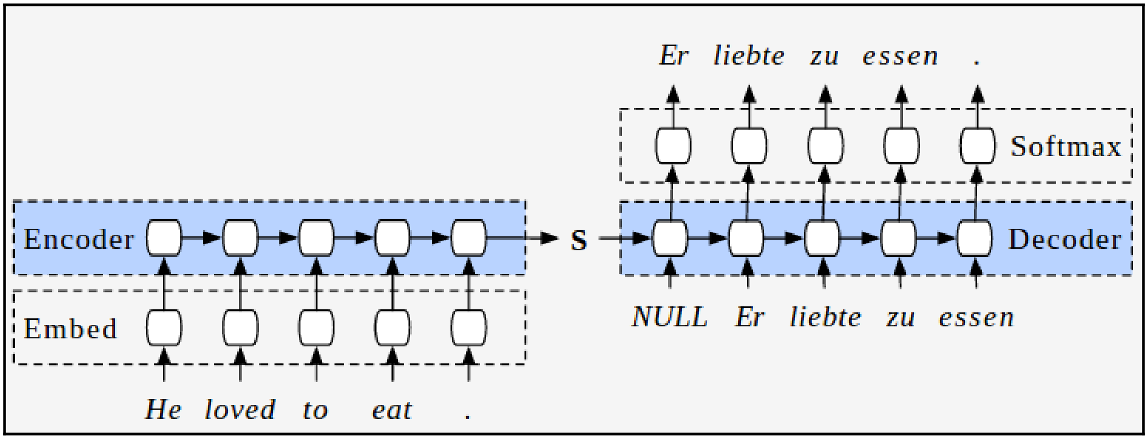


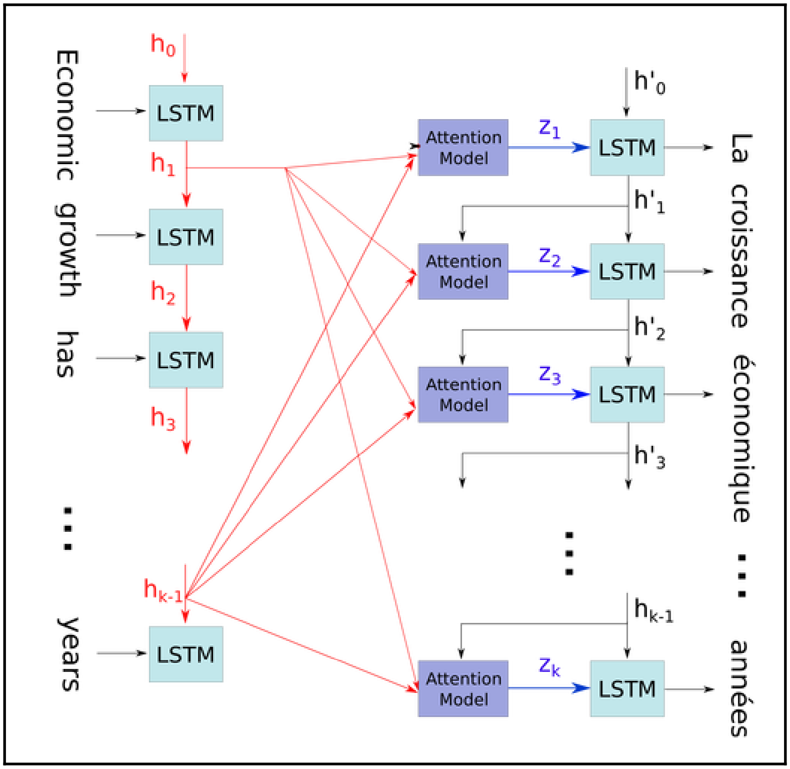
```
# data processing

# data padding
def data_padding(x, y, length = 15):
    for i in range(len(x)):
        x[i] = x[i] + (length - len(x[i])) * [en_word2idx['<pad>']]
        y[i] = [de_word2idx['<go>']] + y[i] + [de_word2idx['<eos>']] + (length-len(y[i])) * [de_
word2idx['<pad>']]

data_padding(X, Y)

# data splitting
X_train,  X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1)

del X
del Y
```

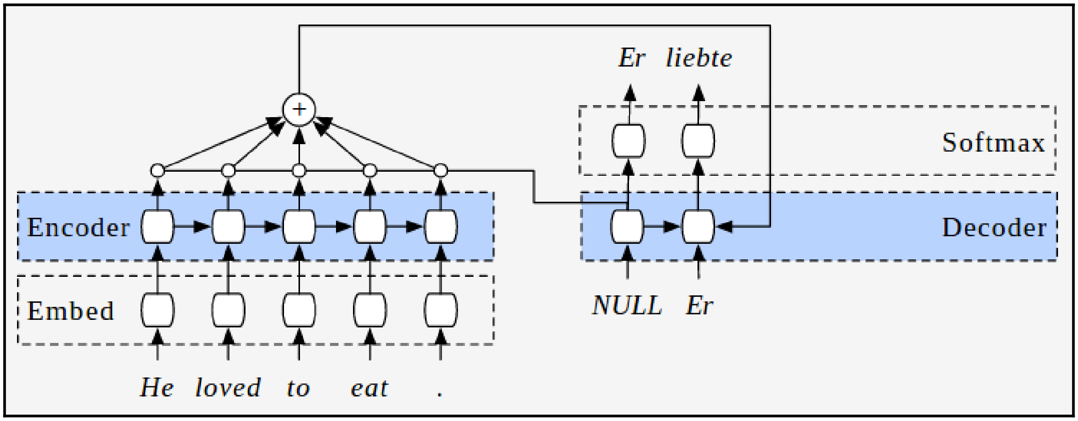


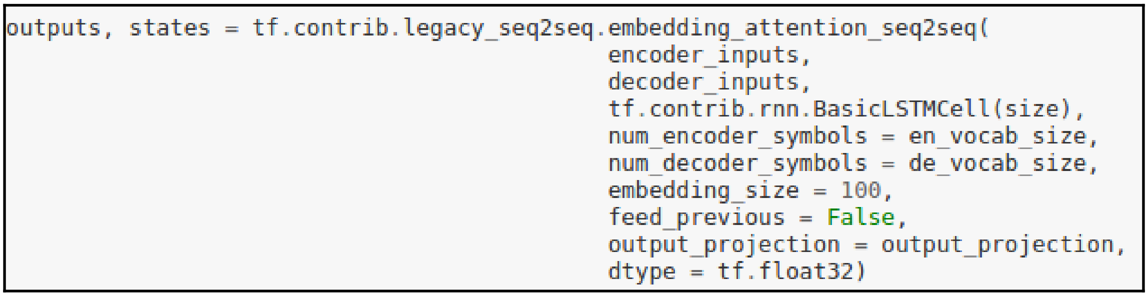
```
input_seq_len = 15
output_seq_len = 17
en_vocab_size = len(en_vocab) + 2 # + <pad>, <ukn>
de_vocab_size = len(de_vocab) + 4 # + <pad>, <ukn>, <eos>, <go>

# placeholders
encoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'encoder{}'.format(i))
 for i in range(input_seq_len)]
decoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'decoder{}'.format(i))
 for i in range(output_seq_len)]

targets = [decoder_inputs[i+1] for i in range(output_seq_len-1)]
# add one more target
targets.append(tf.placeholder(dtype = tf.int32, shape = [None], name = 'last_target'))
target_weights = [tf.placeholder(dtype = tf.float32, shape = [None], name =
'target_w{}'.format(i)) for i in range(output_seq_len)]

# output projection
size = 512
w_t = tf.get_variable('proj_w', [de_vocab_size, size], tf.float32)
b = tf.get_variable('proj_b', [de_vocab_size], tf.float32)
w = tf.transpose(w_t)
output_projection = (w, b)
```



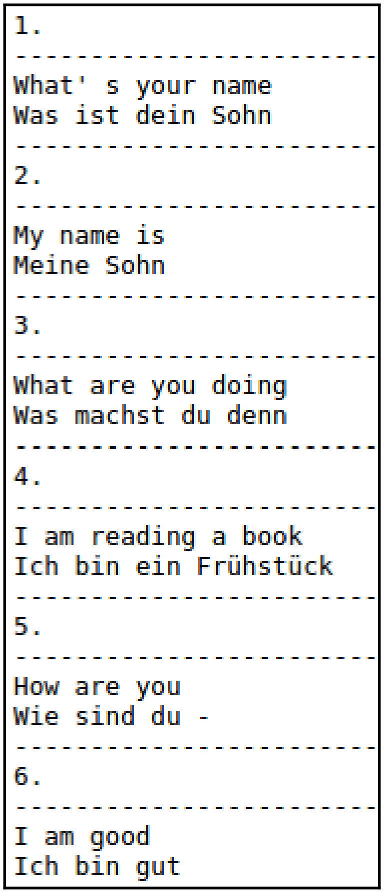

Er liebte zu essen .
Softmax
Encoder
S
Decoder
Embed
NULL Er liebte zu essen
He loved to eat .


Economic growth has ... years
h0
LSTM
h1
LSTM
h2
LSTM
h3
hk-1
LSTM
h'0
Attention Model
z1
LSTM
h'1
Attention Model
z2
LSTM
h'2
Attention Model
z3
LSTM
h'3
hk-1
Attention Model
zk
LSTM
La croissance économique ... années

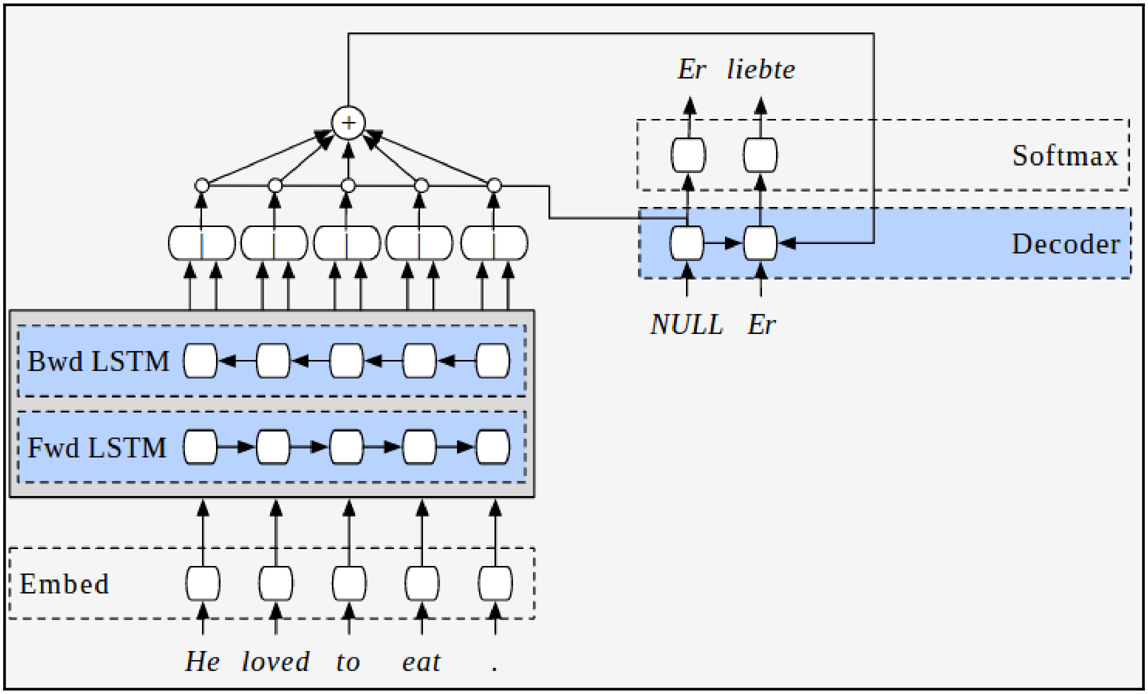


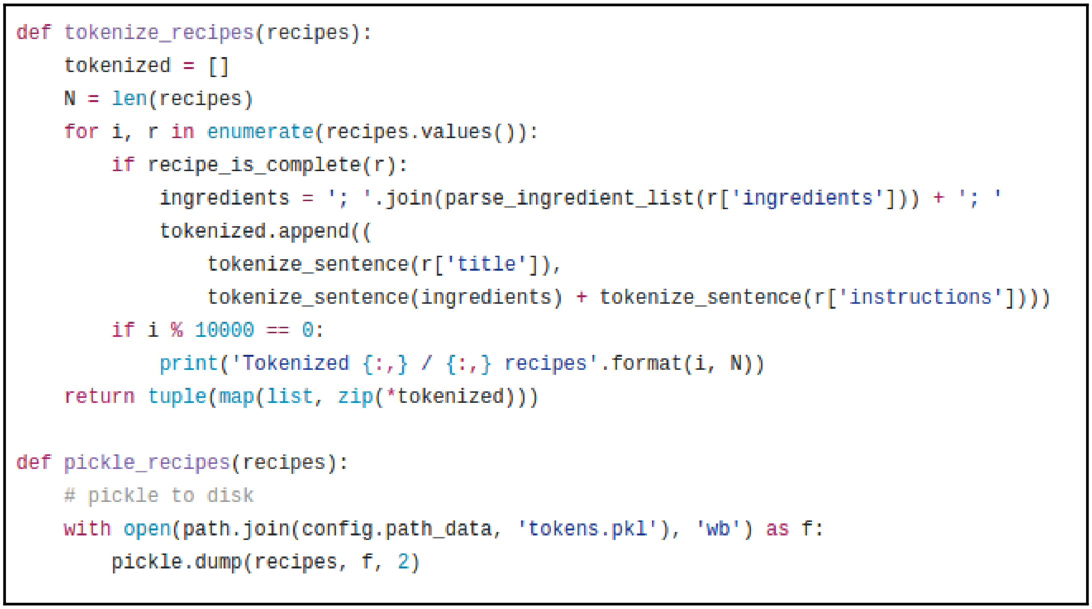


```
outputs, states = tf.contrib.legacy_seq2seq.embedding_attention_seq2seq(
                                            encoder_inputs,
                                            decoder_inputs,
                                            tf.contrib.rnn.BasicLSTMCell(size),
                                            num_encoder_symbols = en_vocab_size,
                                            num_decoder_symbols = de_vocab_size,
                                            embedding_size = 100,
                                            feed_previous = False,
                                            output_projection = output_projection,
                                            dtype = tf.float32)
```

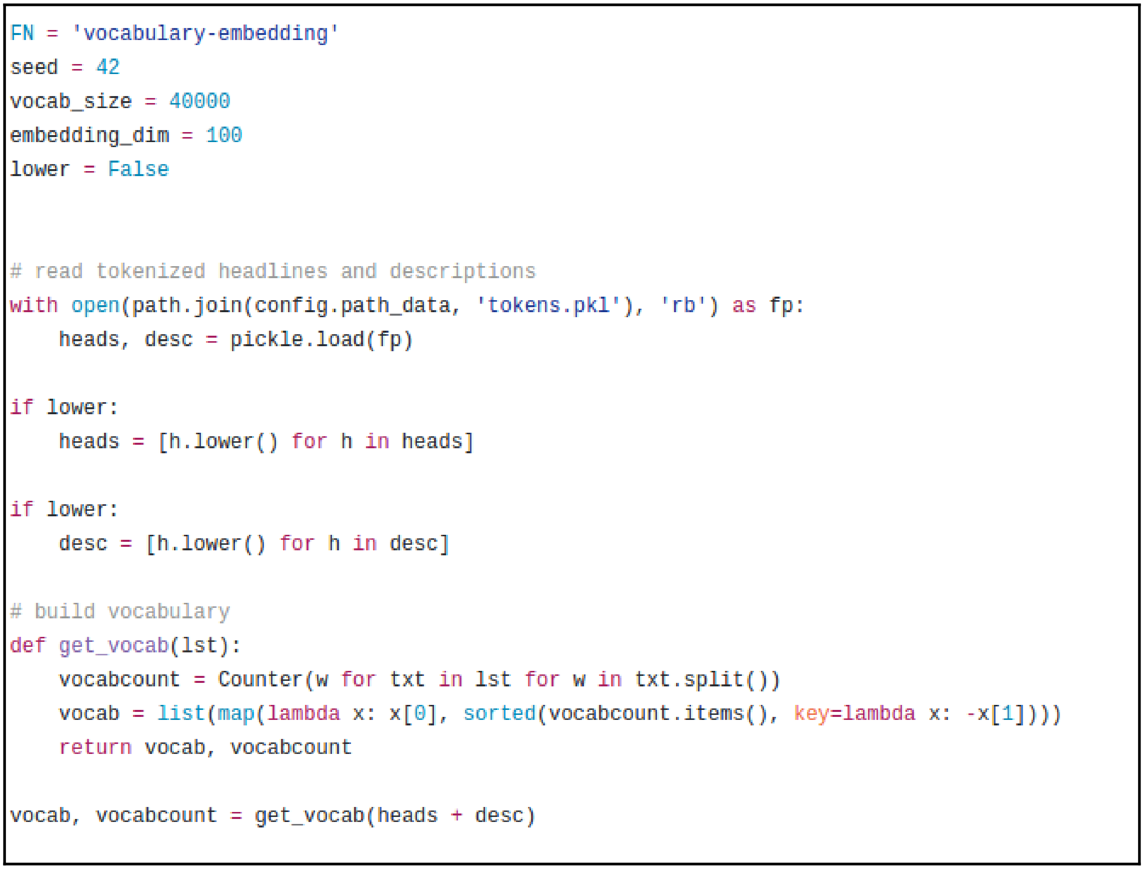


```
1.
------------------------
What' s your name
Was ist dein Sohn
------------------------
2.
------------------------
My name is
Meine Sohn
------------------------
3.
------------------------
What are you doing
Was machst du denn
------------------------
4.
------------------------
I am reading a book
Ich bin ein Frühstück
------------------------
5.
------------------------
How are you
Wie sind du -
------------------------
6.
------------------------
I am good
Ich bin gut
```

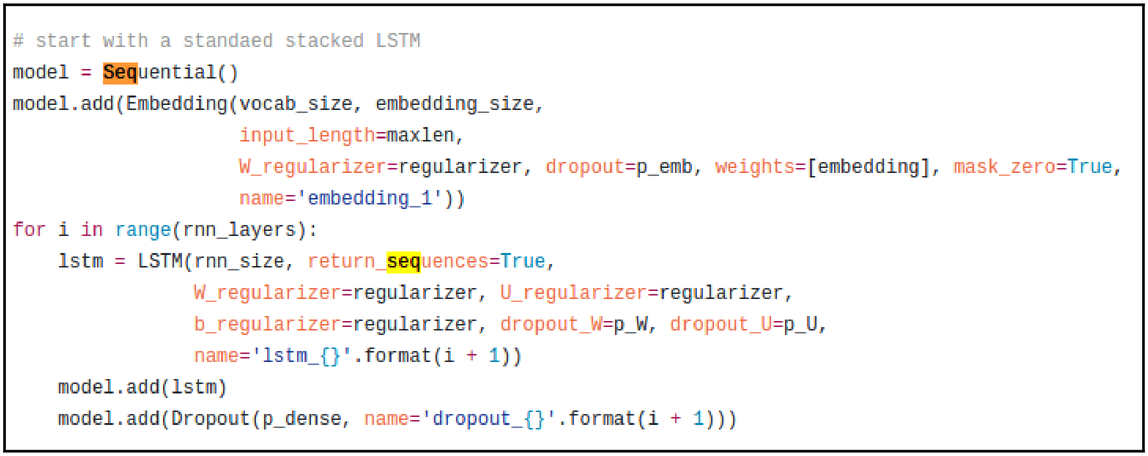




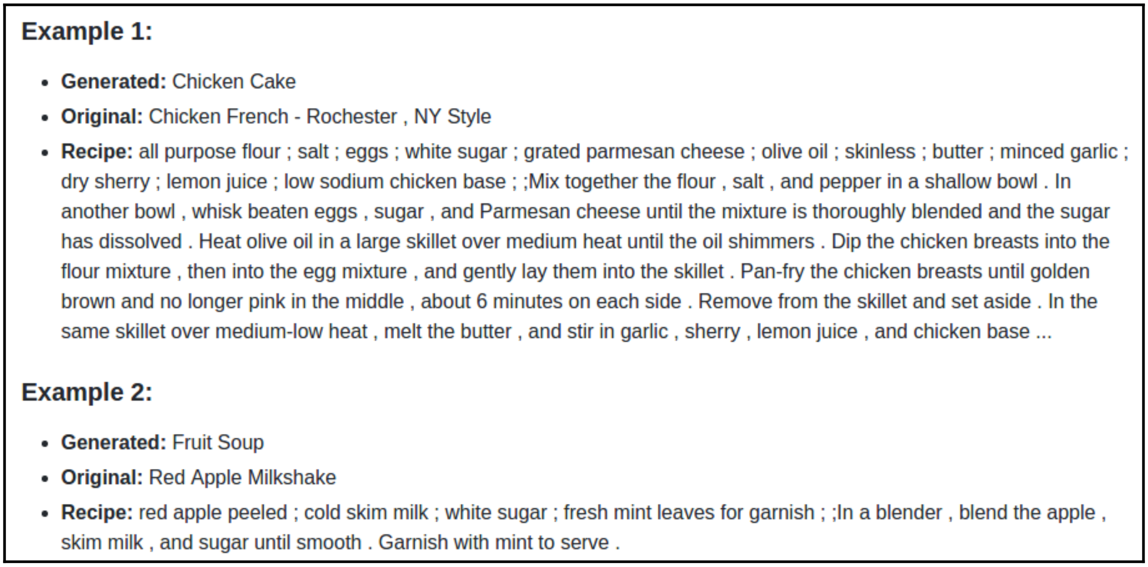
```python
def tokenize_recipes(recipes):
    tokenized = []
    N = len(recipes)
    for i, r in enumerate(recipes.values()):
        if recipe_is_complete(r):
            ingredients = '; '.join(parse_ingredient_list(r['ingredients'])) + '; '
            tokenized.append((
                tokenize_sentence(r['title']),
                tokenize_sentence(ingredients) + tokenize_sentence(r['instructions'])))
        if i % 10000 == 0:
            print('Tokenized {:,} / {:,} recipes'.format(i, N))
    return tuple(map(list, zip(*tokenized)))

def pickle_recipes(recipes):
    # pickle to disk
    with open(path.join(config.path_data, 'tokens.pkl'), 'wb') as f:
        pickle.dump(recipes, f, 2)
```



```
FN = 'vocabulary-embedding'
seed = 42
vocab_size = 40000
embedding_dim = 100
lower = False


# read tokenized headlines and descriptions
with open(path.join(config.path_data, 'tokens.pkl'), 'rb') as fp:
    heads, desc = pickle.load(fp)

if lower:
    heads = [h.lower() for h in heads]

if lower:
    desc = [h.lower() for h in desc]

# build vocabulary
def get_vocab(lst):
    vocabcount = Counter(w for txt in lst for w in txt.split())
    vocab = list(map(lambda x: x[0], sorted(vocabcount.items(), key=lambda x: -x[1])))
    return vocab, vocabcount

vocab, vocabcount = get_vocab(heads + desc)
```



```
# start with a standaed stacked LSTM
model = Sequential()
model.add(Embedding(vocab_size, embedding_size,
                    input_length=maxlen,
                    W_regularizer=regularizer, dropout=p_emb, weights=[embedding], mask_zero=True,
                    name='embedding_1'))
for i in range(rnn_layers):
    lstm = LSTM(rnn_size, return_sequences=True,
                W_regularizer=regularizer, U_regularizer=regularizer,
                b_regularizer=regularizer, dropout_W=p_W, dropout_U=p_U,
                name='lstm_{}'.format(i + 1))
    model.add(lstm)
    model.add(Dropout(p_dense, name='dropout_{}'.format(i + 1)))
```



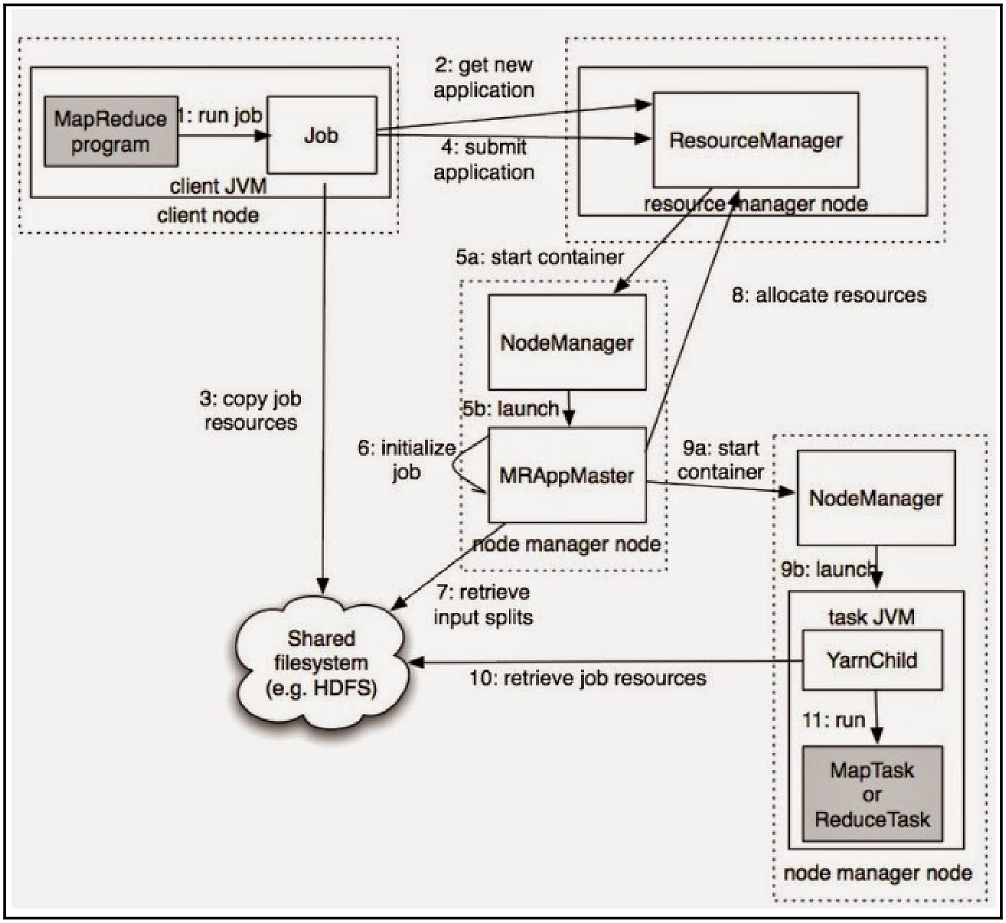
Example 1:

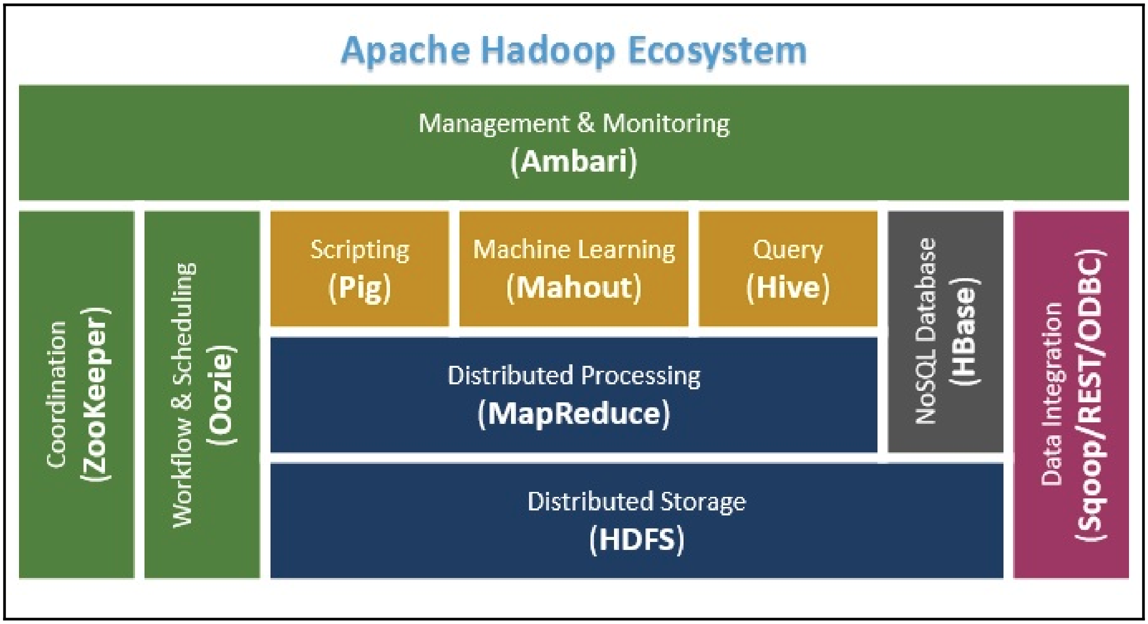
- **Generated:** Chicken Cake
- **Original:** Chicken French - Rochester , NY Style
- **Recipe:** all purpose flour ; salt ; eggs ; white sugar ; grated parmesan cheese ; olive oil ; skinless ; butter ; minced garlic ; dry sherry ; lemon juice ; low sodium chicken base ; ;Mix together the flour , salt , and pepper in a shallow bowl . In another bowl , whisk beaten eggs , sugar , and Parmesan cheese until the mixture is thoroughly blended and the sugar has dissolved . Heat olive oil in a large skillet over medium heat until the oil shimmers . Dip the chicken breasts into the flour mixture , then into the egg mixture , and gently lay them into the skillet . Pan-fry the chicken breasts until golden brown and no longer pink in the middle , about 6 minutes on each side . Remove from the skillet and set aside . In the same skillet over medium-low heat , melt the butter , and stir in garlic , sherry , lemon juice , and chicken base ...

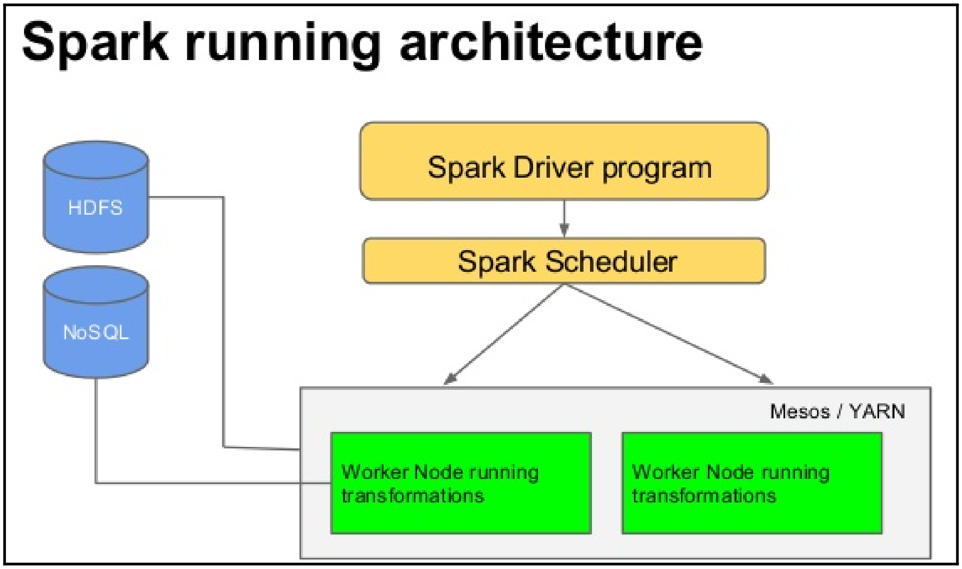
Example 2:

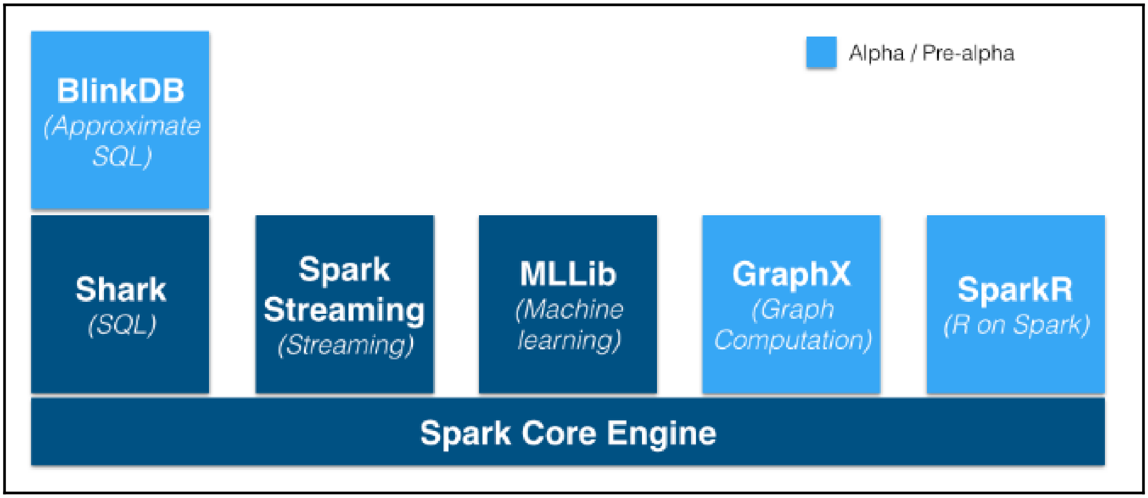
- **Generated:** Fruit Soup
- **Original:** Red Apple Milkshake
- **Recipe:** red apple peeled ; cold skim milk ; white sugar ; fresh mint leaves for garnish ; ;In a blender , blend the apple , skim milk , and sugar until smooth . Garnish with mint to serve .

Chapter 10: Advanced Tools

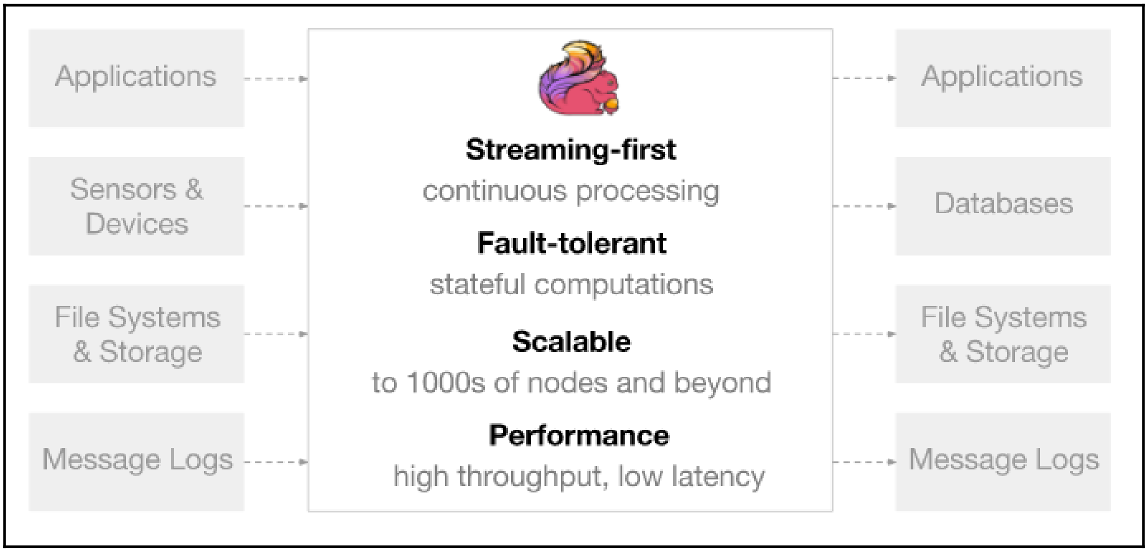



Apache Hadoop Ecosystem
Management & Monitoring
(Ambari)
Coordination
(ZooKeeper)
Workflow & Scheduling
(Oozie)
Scripting
(Pig)
Machine Learning
(Mahout)
Query
(Hive)
NoSQL Database
(HBase)
Data Integration
(Sqoop/REST/ODBC)
Distributed Processing
(MapReduce)
Distributed Storage
(HDFS)


Spark running architecture
HDFS
NoSQL
Spark Driver program
Spark Scheduler
Mesos / YARN
Worker Node running transformations
Worker Node running transformations



BlinkDB
(Approximate SQL)
Alpha / Pre-alpha
Shark
(SQL)
Spark Streaming
(Streaming)
MLLib
(Machine learning)
GraphX
(Graph Computation)
SparkR
(R on Spark)
Spark Core Engine



Applications
Sensors & Devices
File Systems & Storage
Message Logs
Streaming-first
continuous processing
Fault-tolerant
stateful computations
Scalable
to 1000s of nodes and beyond
Performance
high throughput, low latency
Applications
Databases
File Systems & Storage
Message Logs