Link to dataset:
http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Musical_Instruments_5.json.gz

Dataset: Amazon Musical Instruments Reviews

Situation: Data Scientist @ Amazon

Today we will tackling a very common type of EDA/Insight discovery problem, where you've been given a lot of text, and you just have to figure out what is being talked about.

In amazon, where let's say you're a data scientist at, reviews are a very important source of information. Lot's of sellars use it to figure what are the issues customers are facing, how to fix them, how to improve products etc. Amazon wants sellers to improve too, as if thier sales improves, amazon's revenue also goes up.

Problem Statement: what is being talked about in reviews?

You are working in the Musical Instruments department, and you've recently recieved a request to use review data, and figure out **what is being talked about**, and give that insight to sellers so they could possibly improve thier products.

Things to keep in mind:

What do you think a solution to this would look like?

1. A ppt you show to the sellers?
2. A wordcloud of the what is being talked about in the reviews?
3. A list of topics?
4. A list of phrases/words being used?

If you were a sellar what would you be looking forward to?

Each type of solution has it's pros and cons.

1. Some are easy to do, but hard to correctly understand
    1. Example: Wordcloud. Wordcloud is easy to do, but it's very difficult to gain usable insight from it.

2. Some are easy to digest and consume, but hard to do correctly
    1. A list of topics: Very to easy to understand, but how would you generate these topics?

Downloading Data

```
!gdown 1gGazb-hGBiJnVy4WCoWFOFfzPdhlmqrV
```

```
Downloading...
From: https://drive.google.com/uc?id=1gGazb-hGBiJnVy4WCoWFOFfzPdhlmqrV
To: /content/Musical_Instruments_5.json
100% 7.45M/7.45M [00:00<00:00, 87.5MB/s]
```

Data Reading and simple EDA

```
import pandas as pd
import numpy as np

review_data = pd.read_json("Musical_Instruments_5.json", lines=True)
review_data
```

	reviewerID	asin	reviewerName	helpful	reviewText	overall	
0	A2IBPI20UZIR0U	1384719342	cassandra tu "Yeah, well, that's just like, u...	[0, 0]	Not much to write about here, but it does exac...	5	
1	A14VAT5EAX3D9S	1384719342	Jake	[13, 14]	The product does exactly as it should and is q...	5	
2	A195EZSQDW3E21	1384719342	Rick Bennette "Rick Bennette"	[1, 1]	The primary job of this device is to block the...	5	
3	A2C00NNG1ZQQG2	1384719342	RustyBill "Sunday Rocker"	[0, 0]	Nice windscreen protects my MXL mic and preven...	5	W
4	A94QU4C90B1AX	1384719342	SEAN MASLANKA	[0, 0]	This pop filter is great. It looks and perform...	5	
...	...	...	...	...	...	...	
10256	A14B2YH83ZXMPP	B00JBIVXGC	Lonnie M. Adams	[0, 0]	Great, just as expected. Thank to all.	5	

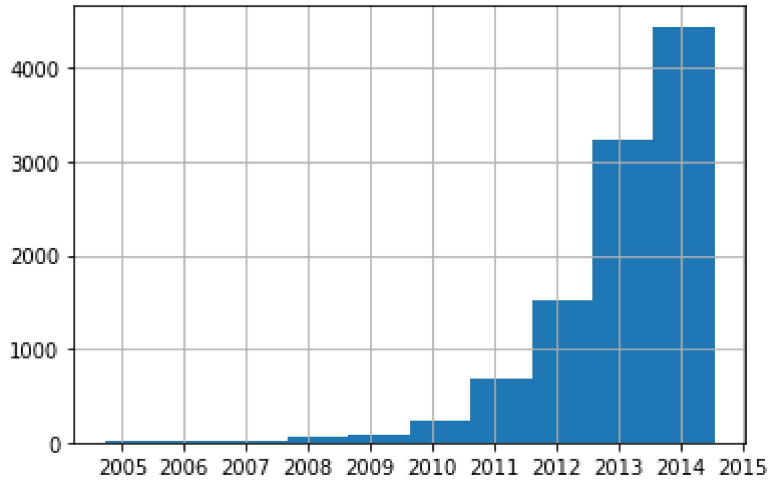
Hmm, the date format looks wierd. Let's fix that first

```
review_data["reviewTime"] = pd.to_datetime(review_data["reviewTime"], format="%m %d, %Y")
```

Looks like we have reviews from 2005-2014, with most of the data coming from recent years.

```
review_data["reviewTime"].hist()
```

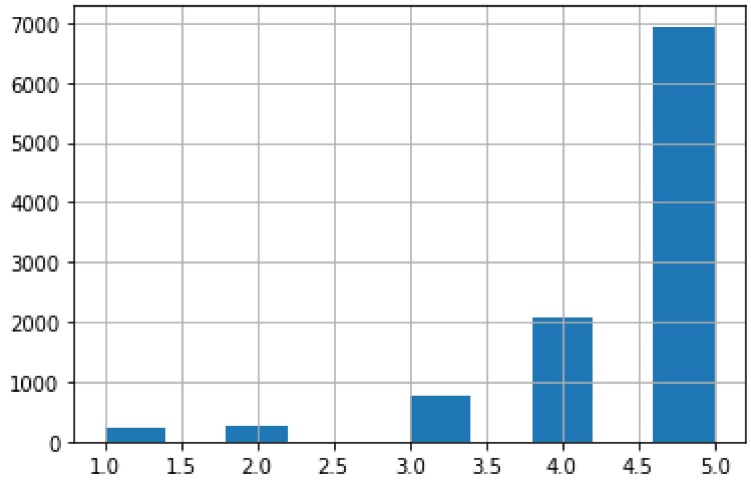
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb87ea38c90>
```



Looking at ratings, most reviews seem to be very positive.

```
review_data["overall"].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb87e9f9510>
```



Since we're not really concerned with the sentiment or ratings, for this problem statement we'll only be focusing on the review content

What if we just count words?

What are the things being talked about in the reviews?

Let's first try a simple word counter, and see the top 10 words?

```
from collections import Counter
def flatten(list_of_lists):
    flat_list = []
    for  l in list_of_lists:
        flat_list.extend(l)
    return flat_list
```

Let's install NLTK, spacy and wordcloud

```
!pip install nltk
!pip install wordcloud
import nltk
nltk.download('all')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.7)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from n
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.7/dist-packa
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Requirement already satisfied: wordcloud in /usr/local/lib/python3.7/dist-packages (1
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fro
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package alpino to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/alpino.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping
[nltk_data]    |       taggers/averaged_perceptron_tagger_ru.zip.
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/basque_grammars.zip.
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/biocreative_ppi.zip.
[nltk_data]    | Downloading package bllip_wsj_no_aux to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping models/bllip_wsj_no_aux.zip.
```

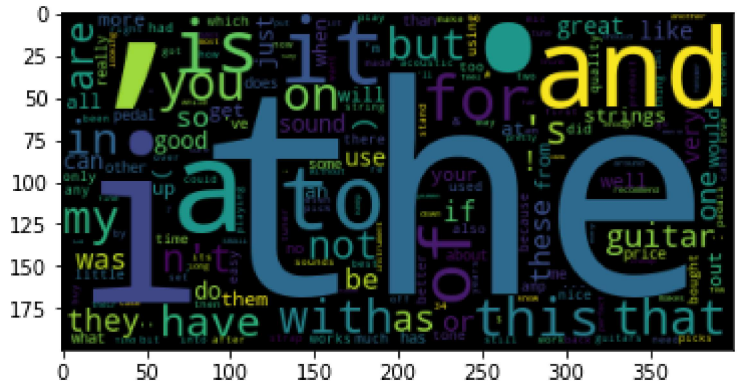
```
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/book_grammars.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package brown_tei to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown_tei.zip.
[nltk_data]    | Downloading package cess_cat to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_cat.zip.
[nltk_data]    | Downloading package cess_esp to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_esp.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/city_database.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
```

```
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud
from matplotlib import pyplot as plt

def wordcloud_plot(counter_all):
    w = WordCloud().generate_from_frequencies(frequencies=dict(counter_all))
    plt.imshow(w)

def tokenize(text):
    tokens = [w for w in word_tokenize(text.lower())]
    return tokens
```

```
counter_all = Counter(flatten(review_data["reviewText"].apply(tokenize).tolist()))
counter_all.most_common(10)
```

```
[('.', 47090),
 ('the', 44092),
 (',', 35705),
 ('i', 30997),
 ('a', 27895),
 ('and', 26984),
 ('it', 24119),
 ('to', 23038),
 ('is', 14943),
 ('of', 13588)]
```

```
wordcloud_plot(counter_all)
```



The 10 most **common words seem to all be stop words** and/or punctuations.

The wordcloud also doesn't look very useful

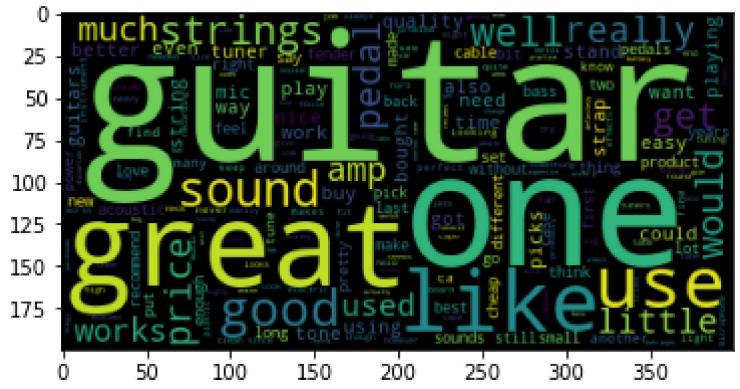
What if we remove stopwords?

Let's try and improve this with a simple stopword and punctuation remover, and using better word tokenization

```
from nltk.corpus import stopwords

english_stopwords = set(stopwords.words('english'))


def tokenize_clean(text):
    tokens = [w for w in word_tokenize(text.lower()) if w not in english_stopwords and w.isa
    return tokens


counter_all = Counter(flatten(review_data["reviewText"].apply(tokenize_clean).tolist()))
counter_all.most_common(10)
```

```
[('guitar', 5472),
 ('one', 4402),
 ('great', 4026),
 ('like', 3897),
 ('use', 3724),
 ('good', 3720),
 ('sound', 3507),
 ('strings', 3349),
 ('well', 3042),
 ('get', 2657)]
```

```
wordcloud_plot(counter_all)
```



Much better now, but still **not very informative**, considering we care about "**what**" people are talking about.

Most of the words seem to be about how someone "liked" something, or something was "good".

The wordcloud also looks much better!

The only useful things we got are "guitar" and "strings"...

```
!pip install gensim
!pip install pyLDAvis
!pip install spacy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-pack
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Collecting pyLDAvis
  Downloading pyLDAvis-3.3.1.tar.gz (1.7 MB)
     |████████████████████████████████| 1.7 MB 6.2 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Collecting sklearn
  Downloading sklearn-0.0.tar.gz (1.1 kB)
Collecting funcy
  Downloading funcy-1.17-py2.py3-none-any.whl (33 kB)
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pandas>=1.2.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from p
Requirement already satisfied: numexpr in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: numpy>=1.20.0 in /usr/local/lib/python3.7/dist-packages
```

```
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-p
Building wheels for collected packages: pyLDAvis, sklearn
  Building wheel for pyLDAvis (PEP 517) ... done
  Created wheel for pyLDAvis: filename=pyLDAvis-3.3.1-py2.py3-none-any.whl size=136898
  Stored in directory: /root/.cache/pip/wheels/c9/21/f6/17bcf2667e8a68532ba2fbf6d5c72f
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl size=1310 sha25
  Stored in directory: /root/.cache/pip/wheels/46/ef/c3/157e41f5ee1372d1be90b09f74f82b
Successfully built pyLDAvis sklearn
Installing collected packages: sklearn, funcy, pyLDAvis
Successfully installed funcy-1.17 pyLDAvis-3.3.1 sklearn-0.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Requirement already satisfied: spacy in /usr/local/lib/python3.7/dist-packages (3.4.2
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.10 in /usr/local/lib/python3.7
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.7/dis
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: thinc<8.2.0,>=8.1.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.7
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (+
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: typing-extensions<4.2.0,>=3.7.4 in /usr/local/lib/pytho
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/d
```

Wait, why don't we just filter in "useful" words?

Can we remove these "not useful" words? We know that all these reviews are in English, can we use some English Grammer concepts here?

What if we extract all Nouns from the reviews?

We can do this using spacy:

First, let's install spacy, and the english models for spacy

```
!pip install spacy
!python -m spacy download en_core_web_sm
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Requirement already satisfied: spacy in /usr/local/lib/python3.7/dist-packages (3.4.2
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: wasabi<1.1.0,>=0.9.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: thinc<8.2.0,>=8.1.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.10 in /usr/local/lib/python3.7
Requirement already satisfied: typing-extensions<4.2.0,>=3.7.4 in /usr/local/lib/pytho
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.7/dis
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.7
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.11.0,>=1.7.4 in /usr/local/lib
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/di
Requirement already satisfied: smart-open<6.0.0,>=5.2.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/l
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packl
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packa
2022-11-07 11:12:55.686249: E tensorflow/stream_executor/cuda/cuda_driver.cc:271] fail
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Collecting en-core-web-sm==3.4.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_
     |████████████████████████████████| 12.8 MB 2.2 MB/s
Requirement already satisfied: spacy<3.5.0,>=3.4.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.7
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: wasabi<1.1.0,>=0.9.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.11.0,>=1.7.4 in /usr/local/lib
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.10 in /usr/local/lib/python3.7
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.7/dis
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: thinc<8.2.0,>=8.1.0 in /usr/local/lib/python3.7/dist-pa
```

```
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dis
```

Now let's load the english model

```
import spacy
nlp = spacy.load("en_core_web_sm")
```

Let's take the first review and use spacy on it

```
review_sample = review_data["reviewText"].iloc[0]
review_sample
```

```
'Not much to write about here, but it does exactly what it's supposed to. filters out t
he pop sounds. now my recordings are much more crisp. it is one of the lowest prices po
p filters on amazon so might as well buy it, they honestly work the same despite their
```

```
spacy_process_review = nlp(review_sample)
spacy_process_review
```

```
Not much to write about here, but it does exactly what it's supposed to. filters out
the pop sounds. now my recordings are much more crisp. it is one of the lowest prices
pop filters on amazon so might as well buy it, they honestly work the same despite
their pricing,
```

From the first glance, it looks similar, except the quotes (") have dissapeared. In fact, after processing, **spacy does NOT return a string object, but a custom spacy Document object!**

```
type(spacy_process_review)
```

```
spacy.tokens.doc.Doc
```

Similarly, **each word in the document isn't a simple string either, but a Token object**! So spacy has done tokenisation and stuff for us already.

Each token object also has a lot of useful properties that we can use.

```
spacy_process_review[0], type(spacy_process_review[0])
```

```
(Not, spacy.tokens.token.Token)
```

Here, we can get the part of speech (i.e. grammer classification) of each word. The word "Not" here, is a particle, denoted by "PART". But let's look for all Nouns!

How are these Part of Speech (POS) tags generated, though? What is spacy doing?

Spacy is using it's english pre-trained model, which was trained using tagged tree-bank data.

This tree-bank dataset contains the POS tag for every word, and these datasets were labelled manually.

1. Model used could be Probabalistic HMM (Hidden Markov Model), which can be trained using a DP algo called Viterbi Algo
2. Model could also be a sequential neural network like RNN (Recurrent Neural Network), trained using backpropogation

Spacy recognises Noun as "NOUN" and proper nouns (names of spceific things like people, cities etc.) as seperate "PROPN"

```
spacy_process_review[0].pos_
```

```
'PART'
```

Awesome! Now we can see the POS tags int this review, we can see that is talking about pop-sounds, recording, prices & filters, by just looking at the Nouns

```
spacy_process_review
```

```
Not much to write about here, but it does exactly what it's supposed to. filters out
the pop sounds. now my recordings are much more crisp. it is one of the lowest prices
pop filters on amazon so might as well buy it, they honestly work the same despite
their pricing,
```

```
for tok in spacy_process_review:
    if tok.pos_ in ["NOUN", "PROPN"]:
        print(tok, tok.pos_)
```

```
pop NOUN
sounds NOUN
recordings NOUN
prices NOUN
filters NOUN
```

```
amazon NOUN
pricing NOUN
```

But we're still missing out on a lot of context, especially like "pop" and "sounds" are actually just parts of ONE thing, "pop sounds".

It's not just the noun words we need... but entire NOUN PHRASE

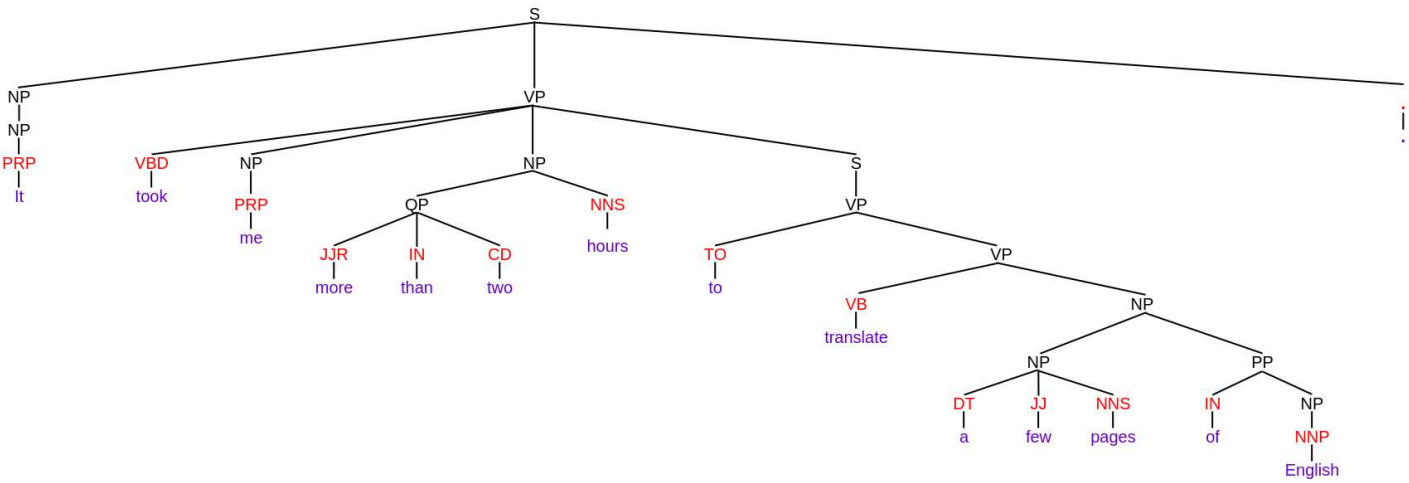
We can achieve this by using constituency parsing.

Constituency parsing uses hand built rules. **These rules are often written usin CFG (Context Free Grammer) and Finite Automoata concepts.**

The rules could be something simple like "NP -> Noun Noun". This basically means that 2 Nouns together, constitutes a Noun Phrase (NP)

In reality though, it would look something like this, and be resolved using CFG rules

```
NP -> Noun
NP -> PropN
NP -> Adj [Adp] NP
```



If you use spacy, though, you don't need to worry about writing your own rules for getting Noun Phrases.

Spacy, treats pronouns as nouns as well.

While useful in many scenarios, we need to remove these noun chunks.

Let's exclude all noun chunks which have no NOUN or PROPN.

Spacy gives the "most expansive", i.e. largest viable noun chunks. This can be very useful in many situation, however ours isn't really one of them. Let's also remove the non-noun parts of the noun-phrase, as our use case doesn't really need that.

```
for c in spacy_process_review.noun_chunks:
    print(c, [t.pos_ for t in c])
```

```
 it ['PRON']
 exactly what ['ADV', 'PRON']
 it ['PRON']
 the pop sounds ['DET', 'NOUN', 'NOUN']
 my recordings ['PRON', 'NOUN']
 it ['PRON']
 the lowest prices ['DET', 'ADJ', 'NOUN']
 filters ['NOUN']
 amazon ['NOUN']
 it ['PRON']
 they ['PRON']
 their pricing ['PRON', 'NOUN']
```

```
noun_checker = lambda x: any()
```

```
for c in spacy_process_review.noun_chunks:
    new_nc = [t.text for t in c if t.pos_ in ["NOUN", "PROPN"]]
    if new_nc:
        print(new_nc)
```

```
 ['pop', 'sounds']
 ['recordings']
 ['prices']
 ['filters']
 ['amazon']
 ['pricing']
```

Looks much better now!

Let's also filter out the non-noun words from the noun-phrase

Before we move on, let's run this on all our reviews once, and see the results

```
def get_noun_phrases(text):
    doc = nlp(text)
    required_nc = []
    for c in doc.noun_chunks:
        new_nc = [t.text for t in c if t.pos_ in ["NOUN", "PROPN"]]
        if new_nc:
            new_nc_text = " ".join(new_nc)
            required_nc.append(new_nc_text)
    return required_nc
```

```
from tqdm.auto import tqdm
tqdm.pandas()
```

```
review_data["noun_phrases"] = review_data["reviewText"].progress_apply(get_noun_phrases)
```

```
100%                                                  10261/10261 [04:42<00:00, 25.86it/s]
```

This looks so much more usable! Now if we run a counter on it again...

```
review_data["noun_phrases"]
```

```
0        [pop sounds, recordings, prices, filters, amaz...
1        [product, bonus, screens, hint, smell, grape c...
2        [job, device, breath, popping sound, voice, re...
3        [MXL mic, pops, thing, gooseneck, screen, posi...
4                [pop filter, studio filter, vocals, pops]
                               ...
10256                                                   []
10257    [Nanoweb strings, while, price, strings, comme...
10258    [strings, past, Elixirs, disconnect, guitar, c...
10259    [Elixir, DEVELOPED, Taylor Guitars, strings, R...
10260    [strings, unwound strings, complaint, strings,...
Name: noun_phrases, Length: 10261, dtype: object
```

Now this looks much better!

It seems like most people are talking about the guitars, strings, the pricing, how they sound, how much amp the devices have, thier tones etc.

Much more useful than the previous iterations!

```
counter_all = Counter(flatten(review_data["noun_phrases"].tolist()))
counter_all.most_common(10)
```

```
[('guitar', 3212),
 ('strings', 2526),
 ('price', 1795),
 ('sound', 1698),
 ('pedal', 1539),
 ('one', 1256),
 ('amp', 1173),
 ('time', 1146),
 ('tone', 1042),
 ('guitars', 984)]
```

But... these still aren't very actionable.

For example, we still don't know what are people talking about regarding guitars? What word/words is it related to?

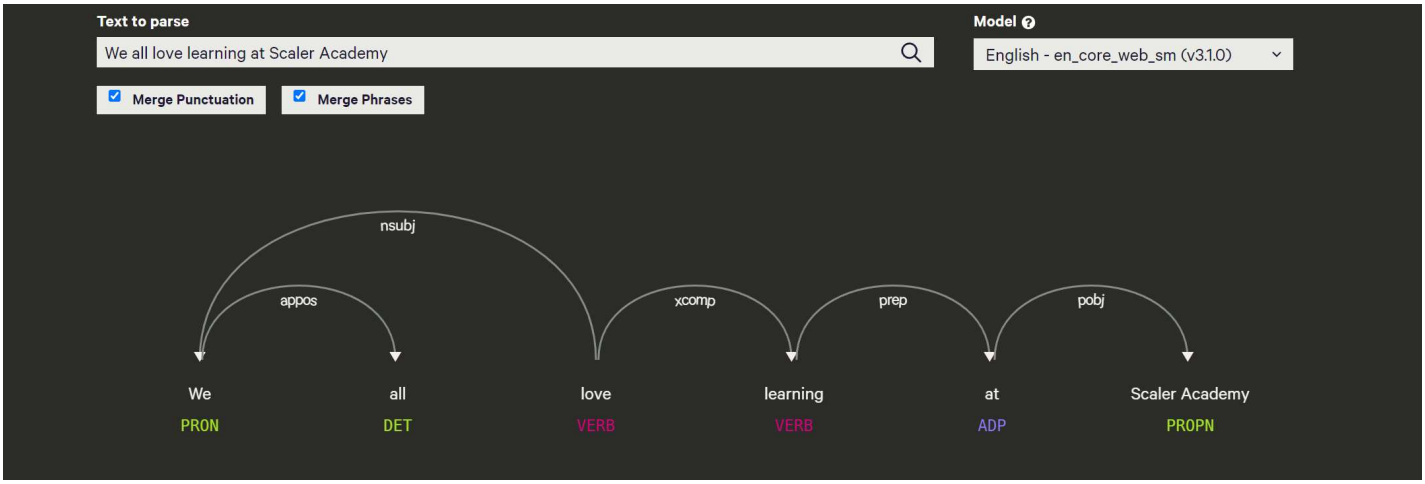
For this, we can use something called as Dependency Parsing.

In Dependecny Parsing, each SENTENCE is converted to a TREE, where every word is connected to some other words based on "grammatical closeness"

Dependency trees can be visualized at (https://explosion.ai/demos/displacy).

Here, the arrows represent a parent-child relationship. Here the is an arrow "love" --> "learning" would imply "love" is the parent of "learning"

Let's take a sample sentence and see it's dependencies.



Here, the word "love" has no arrows pointing towards it, making it the ROOT of the tree, (i.e. it has no parent). The ROOT is almost always the main VERB (action). In this example, "love" is the main action, with "love learning" being the main Verb Phrase.

There is a connection of "love" --nsubj--> "We"

This means "We" is a child of the word "love". In dependency parsing, every parent child relationship has a unique label, which signifies the type of the relationship. This is called as the "dependency type".

Here, the dependecy type is nsubj. nsubj represents the subject or the thing doing the action or the "who" of the action. i.e. Who does "love learning"? "We" does.

Similarly, the "learning" --prep--> "at" has the dependency type "prep" which shows "where". The "at" --pobj--> "Scaler Academy" finished the "where" question. i.e. Where does "love learning" happen? "At". "At" where? "At" "Scaler Academy".

BTW all POS tags and Dependency labels can be found at
https://spacy.io/models/en#en_core_web_sm-labels

Now, let's try to just get the parent of all the noun phrases we got, and attach them?

```
def get_noun_phrases_parent(text):
    doc = nlp(text)
    required_nc = []
    for c in doc.noun_chunks:
        nc_parent = c.root.head
        new_nc = [t.text for t in c if t.pos_ in ["NOUN", "PROPN"]]
        if new_nc:
            new_nc_text = " ".join(new_nc) + "->" + nc_parent.text
            required_nc.append(new_nc_text)
    return required_nc


review_data["noun_phrases_parent"] = review_data["reviewText"].progress_apply(get_noun_phras
```

```
100%                                                  10261/10261 [04:08<00:00, 25.99it/s]
```

```
counter_all = Counter(flatten(review_data["noun_phrases_parent"].tolist()))
counter_all.most_common(10)
```

```
[('price->for', 583),
 ('years->for', 503),
```

```
('guitar->on', 400),
('thing->is', 347),
('strings->are', 278),
('pedal->is', 261),
('guitar->of', 255),
('tune->in', 255),
('job->does', 245),
('strings->of', 242)]
```

Hmm that turned out to not be adding much information for our purpose

We can try more things, for example also looking at at the dependency type of the noun phrase, looking at thier siblings, (i.e. the words that are connected to the parent noun phrase, but not the noun phrase itself).

Also, do notice that the counts are now much lower, earlier "guitar" had occoured nearly 3000 times, but now the most common set is only at 500 times.

Why don't we directly extract the Topic being talked about?

Topic Modelling

While we've had some decent results till now, and have seen some of the most important aspects beeing talked about, i.e. the guitars, strings, the pricing, how they sound, how much amp the devices have, thier tones etc. these stil aren't very general topics.

Maybe we should try to figure out what are topics present in the reviews first?

But what is a topic in the first place?

Let's define a topic as a set of words that are related to each other in soome way.

Example Topic "Animals": "Dog", "Cat", "Bird", "Fish" etc.

But these topics could also be really conceptually complex and specific.

Ex: "Programming Languages used for ML": "Python, R, Scala, Julia"

Thus, any arbitrary set of words can be a topic, as long as they're at least somewhat related to each other.

Also, there is no constraint that one word can only be a part of one topic. One word could be realted to multiple topics.

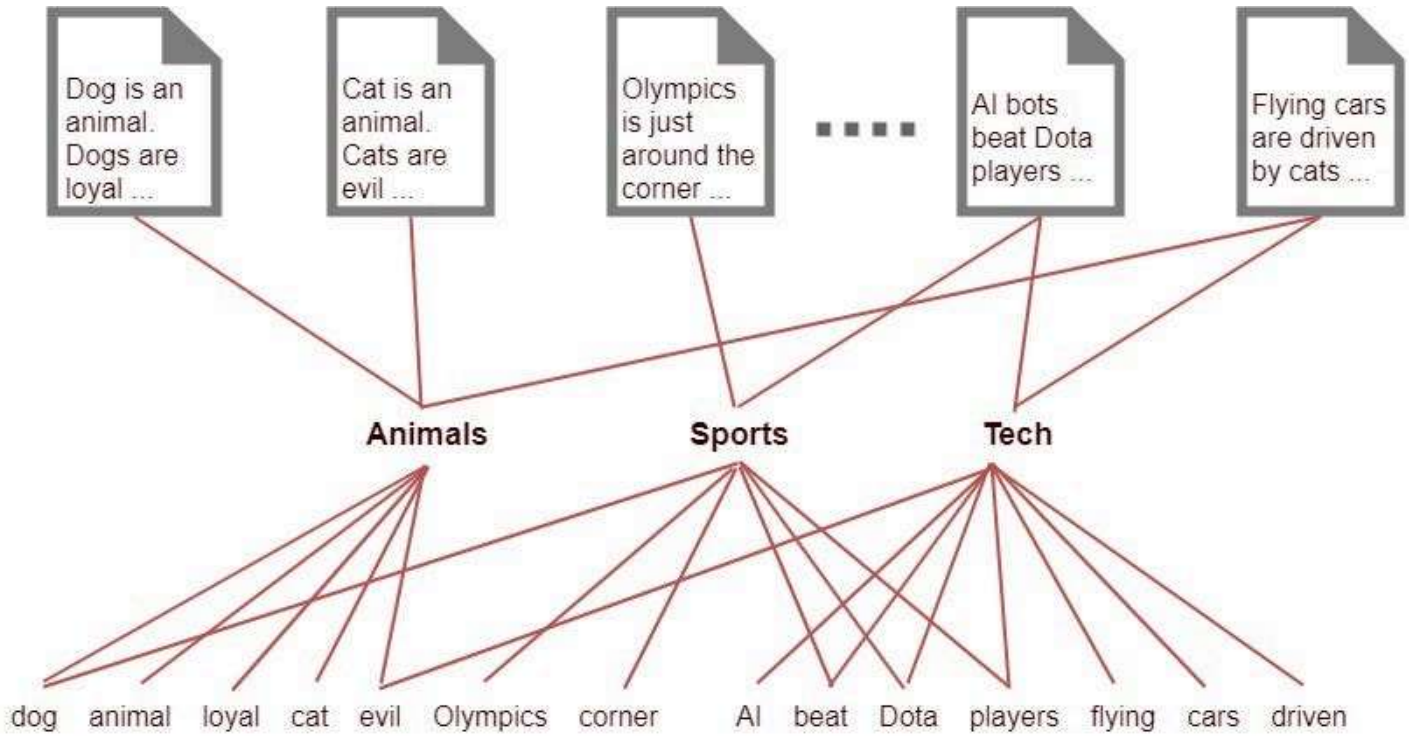
Can we exploit this definition of a topic somehow? What would a review be, in terms of topics?

Could we define a review as a set of topics? Let's say Topics could be: "Guitar", "Amp", "Sound", "Price", "Tone" etc.

Let's first imagine the review as just a bag of words, where order doesn't matter. Could we collect words such that all words would be coming from a topic, as a topic is nothing but a set of words?

If we're able to put 50% of words to "Guitar" topic, 20% to "Amp", could we define the review to be a simple collection of topics?

A review could be defined as something like: 50% "Guitar" + 20% "Amp" + 20% "Sound" + 5% "Price" etc.?



This re-definition of a review/document forms the very basis of a concept called as "Topic Modelling" where we try to find the topics in a set of document, by redifining the documents as a set of topics instead of a set of words.

Let's use the topic definition for directly extracting topics

LDA or Latent Dirichlet Allocation is a topic modelling algorithm, and one of the simplest topic modelling algorithms.

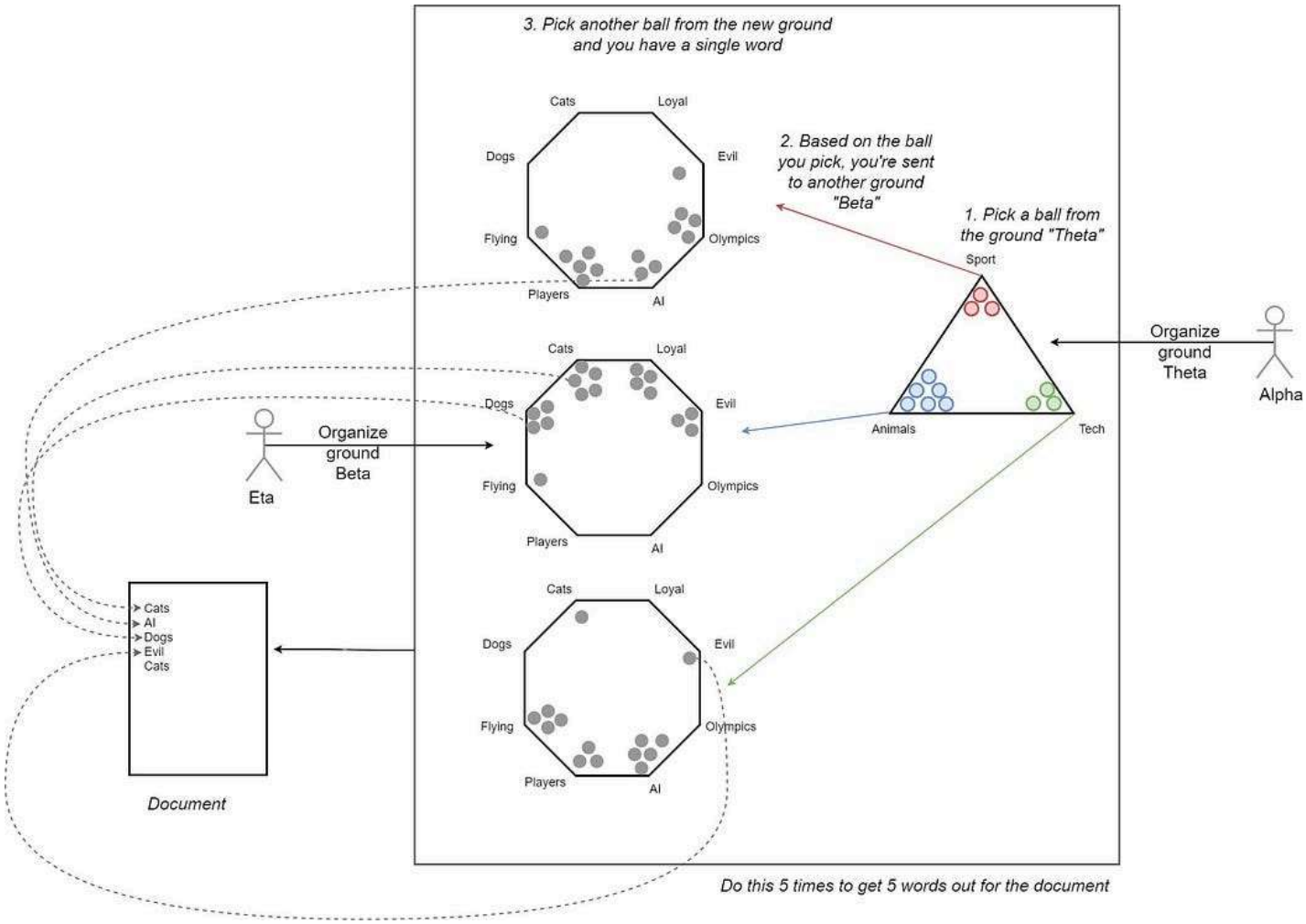
In this, we model this document->topic and topic->word allocation as probability distributions, specifically Dirichlet distributions.

We use 2 hyper-params α and η, which we use to initialize Document-Topic distribution and Topic-word distributions respectively

An analogy of LDA could be like this:

1. Let's say we have to extract 3 topics from a set of documents.
2. Let's imagine we have a basket, corrosponding to which topic goes in what document
    1. For each document, we would pick some balls (say 10) from this topic. Each ball represents a topic.
    2. If say, 8/10 balls belong to **Sport** topic, and 2/10 balls belong to **Animals** topic, we'd say that the document is 80% **Sport** + 20% **Animals**
    3. The "person" who sets and initialises how many balls of each topic into the basket is α, our hyperparamter
3. Let's imagine we have 3 more baskets, each of them corrosponding to one of the topics.
    1. In each basket, we have some balls, with each ball corrosponding to a word. There could be multiple or 0 balls for any word in a basket, and the same word ball can be in several baskets
    2. Each basket can contain different proportions of each ball, example the basket for the topic **Sport** might contain many balls for *Olympics*, *Playing*, *Cricket* etc. But it would probably not contain any ball for *Cats*
    3. The "person" who sets and initialises which balls go into each basket is η, our second hyperparamter
4. We re-imagine the document creation process using these baskets. Recall, that in LDA we consider the document as just a bag of words, without any order.
    1. For every document, we pick up some number of balls (say 10) from the topics basket. Let's say we got 8/10 from **Sport** and 2/10 from **Animals**
    2. We'll then go *8* times to the **Sport** Basket, and pick up some word balls (say 10) each time we go to the basket. We'd also go to *2* times to the **Animals** basket, picking up the same number of word balls for that document.
    3. Now, for the document we have the bag-of-words we'd get using the current topic assigned.

5. We can now compare this generated bag-of-words document to our existing bag-of-words document.
    1. We can now use this comparision to come up with something analagous to a loss function, and optimise the baskets to minimise this loss function.

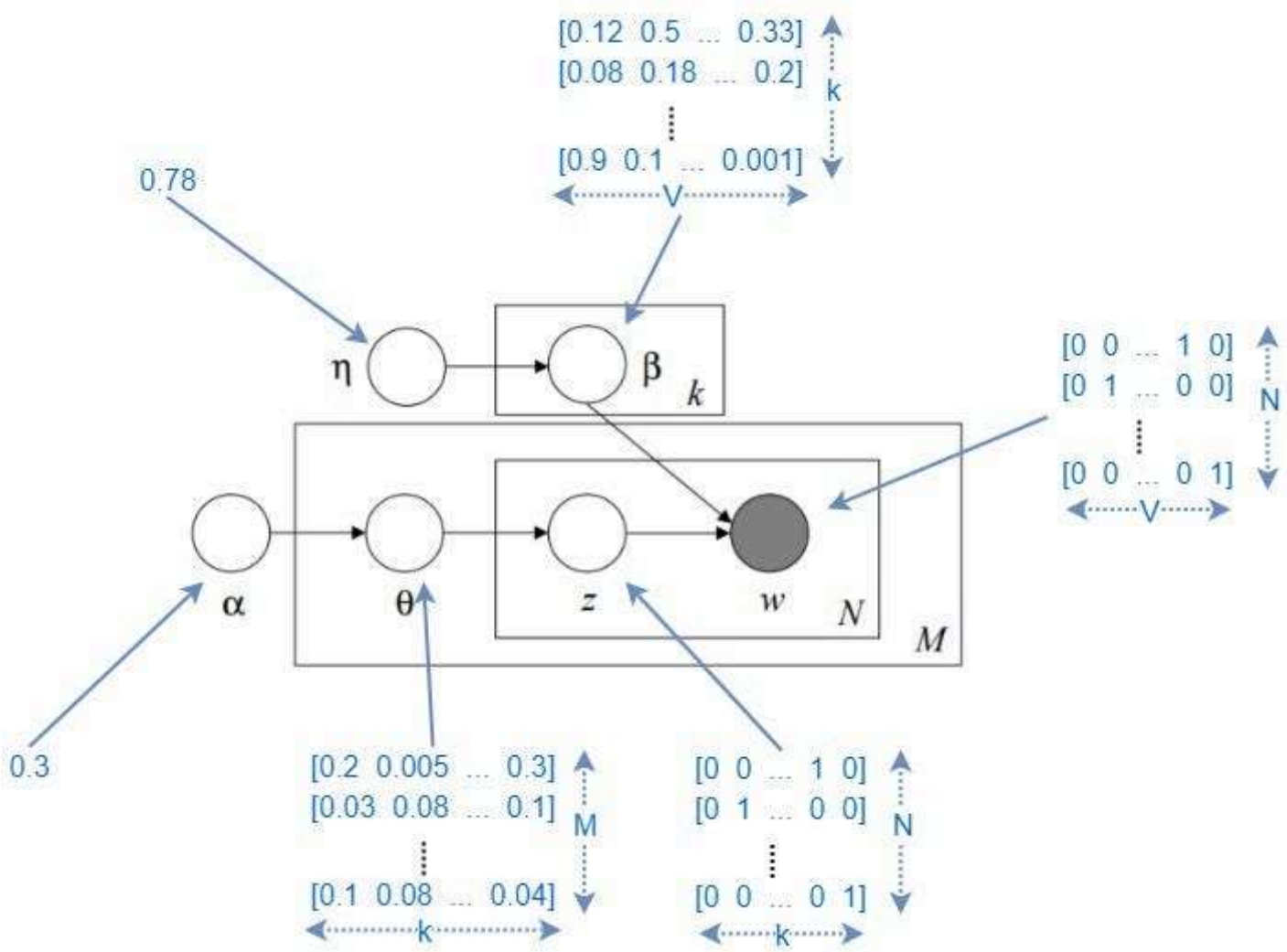


Below is a more technical version of the same algo, for a better understanding of the document generation process Given

- k: Number of Topics
- V: Number of unique words (Vocabulary)
- M: Number of documents
- N: Number of words in each document
- M' : $M \times N \times V$: One-Hot Matrix of Bag of Word Documents
    - M documents, each containing N words, where every one of N words in the document be one of V unique words

Process:

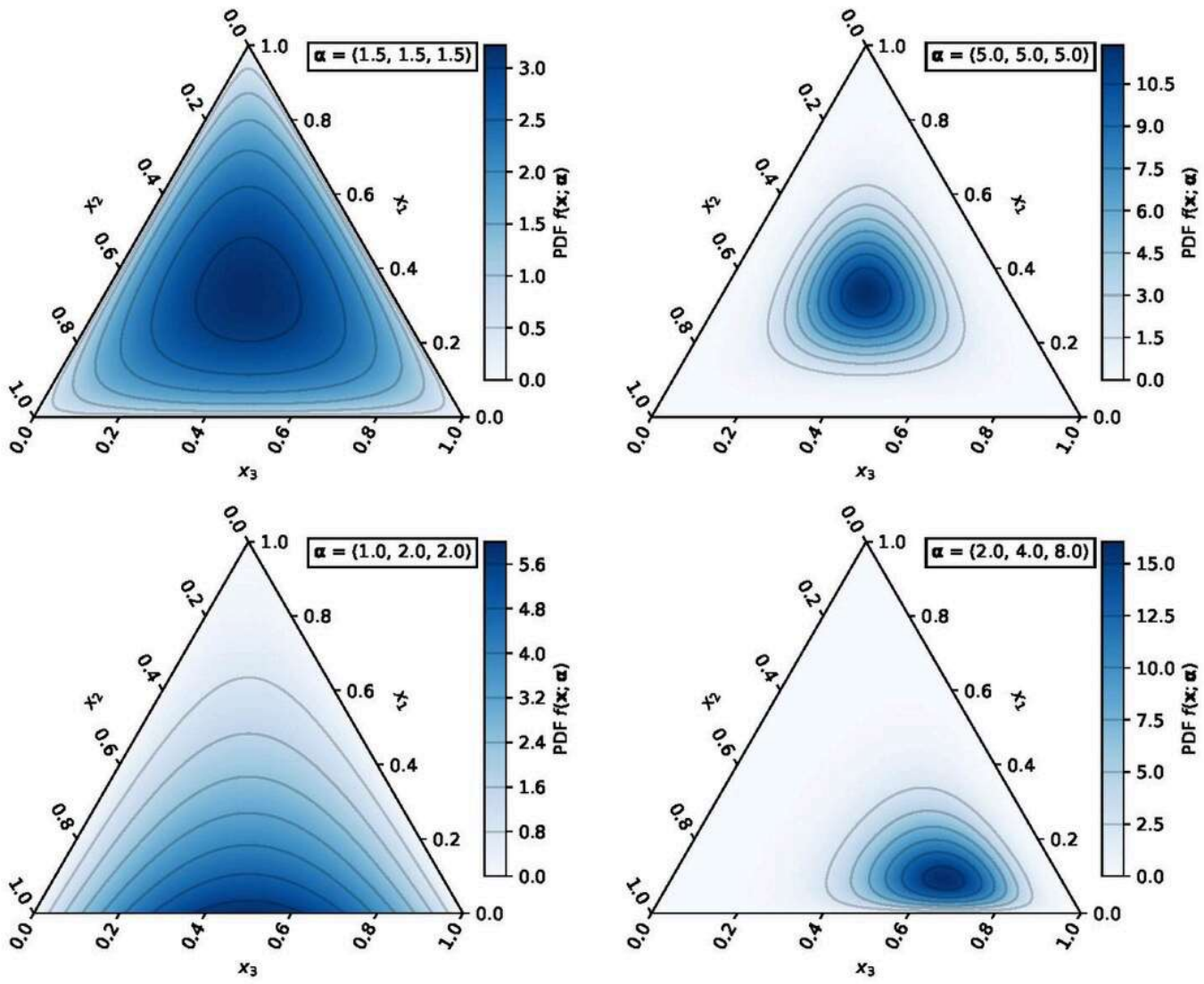
1. We use α to initialize θ. θ is a probablity distribution matrix (size $k \times N$), where every $\theta_{i,j}$ is the probability of topic j being in document i
    1. Similar to sampling from first basket above.
2. For a single document (i.e. one row of $theta$) we sample the topic for each of the N words for the document using it's row in θ. Each word is assigned to a topic, based on the row from θ. We call this One-hot matrix z
3. We also use η to initialize β. β is a probablity distribution matrix (size $k \times V$), where every $\beta_{i,j}$ is the probability of word j being in topic i
    1. Similar to sampling from each baskets above.
4. Now, we can use z and β to sample the bag of words document. For each word in N, we sample the word from β using the topic it was assigned to in z, generating the final document matrix $N \times V$
5. We do steps 2-4 for each document in M, generating a $N \times V$ matrix each time. We finally obtain the $M \times N \times V$ generated documents.
6. We can now compare and optimize this process using the document matrix we already had



For the optimization process, we usually use KL divergence as the Loss function.

For the sampling process, we use the drichlet distribution to get θ from α. Drichlet distribution is a distribution used when you want to get probabilities in a category.

Below are several visulaization of drichlet distribution at different α for 3 topics



Doing ALL of the above in python

Okay... how would you do use this in practice? We can just directly use the `gensim` library's LDA implimentation

```
from gensim.corpora.dictionary import Dictionary
from gensim.models import LdaModel
```

Let's first convert the data into a bag of words document matrix, so we can pass it to the LDAModel.

For this, we'd have to build, the vocabulary, i.e. set of unique words. We can then use this Vocab to convert each document to bag of words.

However, genesim's LDA models don't directly deal with strings, they take list of strings, i.e. pre-tokenized documents.

Let's use the same tokenized and stopword removed, we first used.

```
tokenized_clean_reviews = review_data["reviewText"].apply(tokenize_clean)
```

```
tokenized_clean_reviews
```

```
0        [much, write, exactly, supposed, filters, pop,...
1        [product, exactly, quite, realized, double, sc...
2        [primary, job, device, block, breath, would, o...
3        [nice, windscreen, protects, mxl, mic, prevent...
4        [pop, filter, great, looks, performs, like, st...
                              ...
10256                           [great, expected, thank]
10257    [thinking, trying, nanoweb, strings, bit, put,...
10258    [tried, coated, strings, past, including, elix...
10259    [well, made, elixir, developed, taylor, guitar...
10260    [strings, really, quite, good, would, call, pe...
Name: reviewText, Length: 10261, dtype: object
```

```
vocabulary = Dictionary(tokenized_clean_reviews)
vocabulary
```

```
<gensim.corpora.dictionary.Dictionary at 0x7fb798940fd0>
```

We can then directly use the `doc2bow` of the vocabulary to convert the document to a BOW

```
bag_of_words_documents = [vocabulary.doc2bow(text) for text in tokenized_clean_reviews]
# bag_of_words_documents
```

Let's train for 10 topics now!

```
lda = LdaModel(bag_of_words_documents, num_topics=10, id2word=vocabulary)
```

```
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider ir
```

And let's see the topics.

The number in front of each word represents the *contribution* of each word in that topic.

Looks like the topics are about Mics, Guitars, Guitar Capos, Guitar picks, Tuning, Pedels and Amp etc.

This definetly helps us answer "What is being talked about in the reviews?"!

```
lda.print_topics()
```

```
[(0,
  '0.053*"strings" + 0.017*"sound" + 0.013*"great" + 0.012*"string" + 0.012*"long" +
0.011*"tuner" + 0.011*"guitar" + 0.010*"time" + 0.010*"tone" + 0.009*"good"'),
 (1,
  '0.013*"use" + 0.011*"recording" + 0.010*"cable" + 0.008*"one" + 0.008*"would" +
0.007*"good" + 0.007*"get" + 0.007*"interface" + 0.007*"quality" + 0.007*"device"'),
 (2,
  '0.020*"pedal" + 0.017*"amp" + 0.014*"sound" + 0.012*"tone" + 0.010*"like" +
0.009*"great" + 0.008*"use" + 0.008*"get" + 0.008*"good" + 0.008*"one"'),
 (3
```