Dimensions of Non-Ordinary Experiences: Natural Language Processing of Trascendent Experience Reports

`Senior Thesis [EAS 499]`

`University of Pennsylvania, Fall 2019`

Author

Alex Tianheng Zhao
`alexzhao@seas.upenn.edu`
Department of Computer and Information, School of Engineering and Applied Science, University of Pennsylvania
Department of Statistics, The Wharton School, University of Pennsylvania
`personal website` , `github` , `linkedin` , `ORCID`

Thesis Advisors

Chris Callison-Burch, PhD
`ccb@upenn.edu`
Department of Computer and Information Science (SEAS), University of Pennsylvania

Lyle Ungar, PhD
`ungar@cis.upenn.edu`
Department of Computer and Information Science; additional appoints in the Departments of Bioengineering (SEAS); Genomics and Computational Biology (Penn Medicine); Operations, Informations, and Decisions (Wharton); Psychology (SAS), University of Pennsylvania

Program Coordinator

Max Mintz, PhD
`mintz@cis.upenn.edu`
Coordinator, Senior Thesis Program, Department of Computer and Information Science
University of Pennsylvania

[Cover Page End]

Table of Contents

1. Abstract

Using natural language processing (NLP) techniques on approximately 20,000 reports of transcendent experiences, we examine and better understand the dimensions of non-ordinary states of consciousness. The primary data source is the Vaults of Erowid, which consists in total of approximately 35K experience reports. Generalizable pipelines for machine learning and natural language processing are first discussed, providing a high level overview of the standard processes and tools available to the modern data scientist. We then perform extensive exploratory analysis to understand the data, after which we identify the hidden topics of trip reports through topic modelling, examine clusters of similar experiences, and visualize the dimensions of subjective experiences through word clouds and other techniques. Frameworks for future directions such as predicting substance(s) consumed, trip reports text generation, and feature importance considerations are outlined. Numerous machine learning and NLP techniques are surveyed, and discussed theoretically while noting how they can be applied to future research. Finally, we place this research in historical and academic context, and make a proposal for a more connected and blissful future.

For link to the codebase(which will be continuously updated), which also contains an executable jupyter notebook, see the public project github repository

2. Background and Signifinance

2.1 What are Psychedelics

Psychedelics (from the Greek *psyche*: mind, *delos*: make visible, reveal; together meaning “mind manifesting”), are “powerful psychoactive substances that alter perception and mood and affect numerous cognitive processes” (Nichols, 2016). The most well known “classical psychedelics”, some of which have been used by indigenous cultures for thousands of years, activate the serotonin system, in particular by agonizing the 5HT-2A serotonin receptor subtype (Nichols, 2016). These classical psychedelics include Lysergic Acid Di-amide (LSD, semi-synthetic psychedelic derived from the ergot fungi), Psilocybin (inactive prodrug to the psychoactive Psilocin, found in certain mushrooms), Di-mythyltryptamine (DMT, the active ingredient in the powerful, indigenous Amazonian brew Ayahuasca), and Mescaline (found in Peyote Cactus). Newer psychedelics such as 2-CB, 5-Methoxy-Dimethyltryptamine (5-Meo-DMT, found in multiple plant species and most famously the Sonoran Desert Toad) are also often classified under the term "hallucinogen" or "psychedelic". Other related drugs such as 3,4-Methylenedioxymethamphetamine (MDMA) are often referred to as psychedelic drugs, but are more correctly classified as an empathogen or entactogen (Nicholas, 2016) (meaning “touching within”, from the Greek *en*: within and the Latin *tactus*: touch and the Greek -gen: produce)

2.2 Motivation: Why This Thesis Exists

On September 4th, 2019, Johns Hopkins University launched the Johns Hopkins Center for Psychedelics and Consciousness Research, with $17 million dollars in research funding (Johns Hopkins University, 2019). The center is thought to be the largest of its kind in the world, and sets to study addition, PTSD, addiction, depression, anorexia nervosa, trauma, and other mental “illnesses” with psychedelics and traditional forms of therapy.

After decades of suppression and moral hazard, psychedelics are making a return to legitimate research as a novel and surprisingly effective treatment modality for multiple extremely difficult conditions, including but not limited to end of life depression and anxiety (Griffiths et al., 2016; Osório et al., 2015), treatment-resistant depression (Carhart-Harris et al., 2017), post traumatic stress disorder (PTSD) (Mithoefer et al., 2011; Doblin, 2002), alcohol and nicotine addiction (Nichols, 2016), cluster headaches (Sewell, Halpern, & Pope, 2006), obsessive compulsive disorder (Moreno, Wiegand, Taitano, & Delgado, 2006), and shows early promise for sexual and racial trauma and alzheimer’s. Psychedelics have also shown dramatic results for “well-people”, including being able to reliably induce “mystical-type experiences” (R. R. Griffiths, Richards, McCann, & Jesse, 2006), increase creativity (anecdotally, see Nichols, 2016), and is correlated with greatly enhanced feelings of religious devotion (Pahnke, 1963). Beyond clinical studies, animal studies with octopi have demonstrated MDMA increasing feelings of social connectedness (Edsinger & Dölen, 2018). In vitro studies have shown enhanced neuroplasticity (Victor Acero, preprint). Brain imaging studies, primarily out of Imperial College London and University College London, have shown psychedelics quiet the default mode network (R. L. Carhart-Harris et al., 2012), induce hyperconnected and entropic brain states (Robin L. Carhart-Harris et al., 2014), and proposed a mechanism for action for psychedelics, formalized as the Free Energy Principle and RHEBUS (R. L. Carhart-Harris & Friston, 2019). Selena Atasoy et. al of Oxford University has also recently demonstrated that the brain under psychedelics has more high energy resonant states, while remaining hyper-coherent and highly orchestrated, using a novel technique called Connectome Specific Harmonic Waves (CSHW) (Atasoy, Donnelly, & Pearson, 2016). The Qualia Research Institute (QRI) has also recently proposed the Symmetry Theory of Valence (STV), which posits that symmetries in resonant brain states as modelled by a high dimensional mathematical objects, is correlated with experiential valence (Johnson, 2016).

With all the exciting work being done in the psychedelics landscape (call it “psychedemia”), be it clinical, neuroscientific, or anthropological, little work has been done on understanding what the psychedelic trip actually consists of, and understanding qualitatively and quantitatively the dimensions of the psychedelic subjective experience. Current work by Imperial College and Oxford (pending publication) are showing promising results for understanding both the macro and micro phenomenology of psychedelic experiences. Using natural language as an inadequately wanting proxy, we can begin to understand the hallmark subjective signatures of different substances, and potentially identify novel neuro-targets for clinical interventions to reduce suffering, as well as gain a deeper understanding of mechanisms of action for psychedelic substances, if such a mechanistic understanding is even possible. It is the sincere hope that through this work, we may begin to demystify the subjective dimensions that characterize psychedelic experiences, and lay the foundations for more informed clinical and societal support structures that may alleviate unnecessary mental suffering and enrich the lives of well-people.

This thesis seeks to extend the current work by using techniques from machine learning, and in particular natural language processing, on a corpus of psychedelic trip reports retrieved from multiple sources, including chiefly, the Vaults of Erowid. We will present the results factually, with minimal layers of interpretation and meticulous documentation for high reproducibility, clarifying our assumptions and processes whenever appropriate. It is our hope that this work may be open-sourced and somehow beneficial to clinical researchers, psychologists, neuroscientists, machine learning engineers, social scientists, and/or the lay people community as a whole. If not for some deep, metaphysical takeaway, perhaps for some lighthearted, jovial pass time.

It should also be noted that this thesis focuses heavily on the theoretical foundations of select natural language processing techniques, and demonstrates (not necessarily very deeply) the potential applications of these techniques to a psychedelic corpora. I am a firm believer in understanding things deeply in order to explain them simly, and with solid theoretical foundations, applications arise naturally, and various techniques can be mixed and matched as long as the assumptions are clear.

2.3 Hacky Origins: The Real Reason Why This Thesis Exists

In the spring of 2019, Victor Acero, Rahul Sood, Margaret < last name omitted > and I cofounded the Penn Society for Psychedelic

Science (PSPS) and scrambled together the inaugural Intercollegiate Psychedelics Summit, IPS 2019, at the University of Pennsylvania. IPS 2019 drew about 150 students and 8 speakers, including Frederick Barrett and Manoj Doss from JHU's Center for Psychedeelics and Consciousness Research (Psychedelics Research Unit at the time) and David Nutt, president of the European Brain Council and researcher at Imperial College London's Psychedelics Research Unit. Feeling slightly burnt out from planning the summit in 4 months, we took the summer to recharge, with Victor Acero and Emily Cribas (now PSPS Operational Director) hosting 2 journal clubs over the summer and the remaining cofounders in different corners of the world, recuperating from a very intense 4 months.

Come fall of 2019, we reconvened with great enthusiasm, and collectively ratified PSPS's constitution and bylaws, estalished the **Board of Directors** structure, and welcomed 6 new team members onto the **PSPS Council**, which supports the PSPS General Member Body as a whole. One of the new members was **Yev Barkalov** (`@yevbar`), who is one of the most talented software engineers and self-proclaimed hackers I know, having founded **Hack the Fog** (San Francisco's first high school hackathon) and attended 30+ hackathons just for fun. Around late September, we began working on establishing PSPSLabs, which is an innovative lab for psychedelic projects within the umbrella of PSPS. Think GoogleX, but for psychedelics and consciousness related projects. We were setting up PSPSLabs to provide a nest for potentially impactful, consciousness expanding projects to grow and blossom, and our hope was that through our work, we may enrich the experiences of those around us in safe and responsible ways. With the ethos of silicon valley and a deep, humbling belief that the responsible marrying of technology and science can potentially bring positivity and value to those around us, we began brainstorming projects we could work on for fun to share with the world. Yev and I began calling regularly for hours at a time, sharing resources, chatting about the recent developments in psychedelia, and coming up with random ideas.

One of the projects that blossomed into our shared consciousness on one such call was **EveryQualia**, a search engine for psychedelic trip reports. We were super curious what the experiences of others were like on psychedelic drugs. We began looking at the Johns Hopkins and Imperial College London survey studies, which had super interesting findings such as how the subjective rating of ego dissolution correlated with the quieting of the default mode network (R. L. Carhart-Harris et al., 2012), and how a single high dose psilocyin session let to significant decreaes in depressive symptons (as measured by survey results). After looking at a few studies, we began wondering — is there more out there? The surveys were good because they were rigorous and comparable between subjects, but a few numbers and scales could hardly capture the ineffability, noetic quality, and richness of psychedelic experiences! We became extremely fasincated by free style trip reports, unburdened by the limiting and valve reducing nature of surveys. Alas, after poking around for a while, we realized that there was no good search engine for psychedelic trip reports! There were various reddit threads and personal stories scattered here and there but no centralized data source. Then we found Erowid, which contained a massive archive of about 35,000 trip reports (as of Dec 9, 2019). The Vaults of Erowid was a treasure trove of information. Indeed, Erowid is one of the oldest and most trusted sources for information about hundreds and thousands of drugs, psychedelics and otherwise. We thought we could start by scraping Erowid and saving the trip reports to a database, build a simple webapp frontend, and use something like Algolia (search as a service API) to provide smooth realtime search functionality of the massive Erowid trip report. And just like that, EveryQualia V0 was born, live with about 10K trip reports (we didn't have any money, and Algolia was limited to 10K data objects. Also: if Earth and Fire Erowid are reading this, we apologize for bombarding your serves with our scrapers, and note we totally deserved to be blocked).

Yev became the driving technological powerhouse behind EveryQualia while I dipped deeper into Machine Learning and Natural Language Processing techniques, having taken `CIS 350: Computational Linguistics` with Chris Callison-Burch in the spring of 2019 and now taking `CIS 520: Machine Learning` with Lyle Ungar. With version 0 of our webapp, we had real time search and a groovy interface to read trip reports, which was awesome, but we still didn't have a way to visualize any large scale patterns that were present in the corpus. Since machine learning (ML) is all about learning generalizable patterns from massive amounts of data, applying ML and in particular NLP techniques psychedelics report corpus seemed like a natural extension. Not only would writing this thesis deepen my understanding of the theoretical foundations of machine learning, it also felt like a valuable launching pad for NLP technology that we plan to integrate into EveryQualia. I will say, at this point in a time, just how grateful I am to have two mentors and past professors to sign on as advisors to this rather unconventional thesis. Thank you.

2.4 Previous Work

A research team of the Chemical Youth Project at University of Amsterdam published a fantastic visualization in 2016. This project analyzed 20,534 reports, focusing on a select subset of substances, and analyzed the demographics, language, sustance co-use, and dosage trends in the Erowid Corpus. As of Dec 9, 2019, their website is not live for unkonwn reasons, but a July 23, 2019 archived version can be found here. The visulizations done by this team is absolutely stellar, and the reader is invited to check out their work before reading this thesis to get a better feel for the Erowid Dataset.

2.5 Intended Audience

There is no single mold of an audience for which this thesis is written. A reader who is familiar with the basics of linear algebra and statistics, in particular the basic concepts and notations of vectors, matrices, and probability distributions will find the thesis straighforward to follow especially at times when there are slight conceptual leaps. A few sections contain more advanced probability theory, but those can be skipped without sacrificing the overall narrative structure of the thesis. No prior knowledge is assumed about techniques in machine learning and natural language processing (NLP), as the foundations will be presented from first principles by first relying on intuition and supported by mathematics for those interested. The final sections of the thesis

consists of musings, extensions, and bonus content for the curious reader, and contain a few non-technical elaborations intertwined with technical notes.

This thesis was as much as an empirical exercise as it was a theoretical exercise, written in a rather didactic manner as an self-exercise to test my own understanding of the material and ability to convey not only what was done but also the theory of why what was done is appropriate.

Perhaps more important than prior knowledge for the reader is an innate curiosity, a child-like delight in exploring new frontiers; one who is drawn to principles of foundational interconnectedness and not afraid to embrace concepts that seem at first challenging for the unitiated, yet delightful for the persistent, may find this thesis enjoyable.

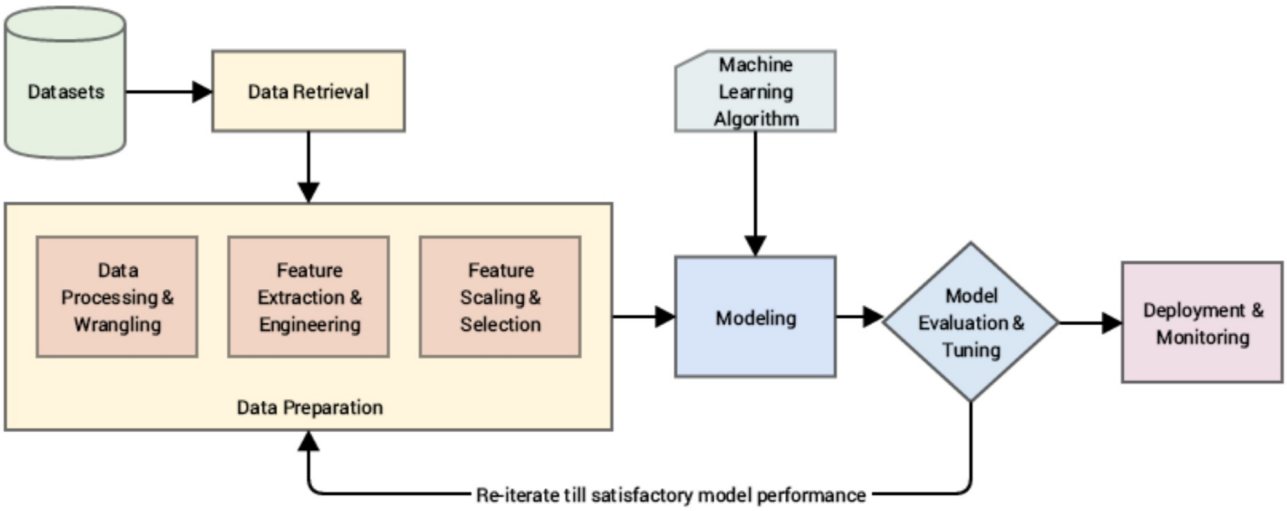
A note: due to scope and time constraints, a few state of the art techniques such as finetuning BERT and GPT 2 language models were researched and planned, but not fully explored. I have left those sections in the table of contents, a deference to what was conceived, and a gentle reminder to future self that even a thesis is a work in progress and doesn't have to be absolutely "perfect".

2.5 The Machine Learning Pipeline

The field of machine learning is very broad, and indeed a complete overview of what machine learning would fill a complete tome and certainly out of scope for this thesis. That being said, there are a few generalizable insights and principles that are very important to mention. They are highly inspired by Professor Pedro Domingos's excellent paper, A Few Useful Things to Know about Machine Learning and my work in progress computer science and statistics education

- machine learning is about generalizing from examples
- machine learning algorithms take *data* as inputs, and attempt to find *patterns* in the data, in order to accomplish some task(s) of interest
- There are a few key types of problems in machine learning, and often the first step to translating theory into practice is recognizing the type of problem we are trying so solve: (1) **Regression** problems are about predicting *continuous* outcomes **y** from a collection of continuous or categorical features **X**, such as predicting housing prices (**y**) from [square footage (continuous feature), and whether it's in a good neighborhood (categorical feature)] (**X**); Regression is a *supervised* problem because it requires labelled training data **y** (2) **Classification** problems are about predicting discrete class labels **y** from a collection of continuous or categorical features **X**, such as classifying whether a person has a heart condition (discrete **y**) based on [blood pressure (continuous feature), and smoking habits (discrete feature)]; Classification is a *supervised* problem because it requires labelled training data **y** (3) **Clustering** problems are about finding clusters of observations that are most “similar” to each other, such as finding customers who are similar, or which groups (documents, reports, abstract entities etc.) are most closely related to each other; Clustering is an *unsupervised* problem because it doesn't require labelled training data **y**: the patterns are discovered directly from the observations.

This thesis employs the CRISP-DM (Cross Industry Standard Process for Data Mining) pipeline, which is a widely used open standard by professional data scientists as a standard data mining approach



A standard machine learning pipeline (source: Practical Machine Learning with Python, Apress/Springer)

`CRISP-DM Model Infographic Source`

The algorithms chosen in the *Modelling* stage can significally impact conclusions. Many papers use black box algorithms and present results as if they are facts, but different initial conditions and design decisions can very quickly change "facts" into approximations. Indeed, machine learning is all about being approximately sure of things, an intuition formalized by probability distributions and statistical esimtators.

3. Data Acquisition and Origins

The first step to any machine learning or natural language processing project is to acquire the data. In this section, we describe the data source and data scraping process.

3.0 Switching Between Local Jupyter and Colab Development Environments

N.B.: This codeblock exists to switch between Jupyter and Google Colab developement environments, and is applicable only to readers who wish to run the notebook. Change the `root_path` variable to the appropriate path in order to reproduce the code. Additional work may be required to run the code without glitches. Readers are invited to submit pull requests to contribute to the codebase.

In [8]:

```
# toggle between local and Google Colab Development Environment
# Uncomment below if using colab environment
# from google.colab import drive
# drive.mount('/content/gdrive')

## if developing locally, set is_colab to False
# is_colab = False
# root_path = '/content/gdrive/My Drive/UPenn/UPenn_fall-2019/[EAS 499] Senior Thesis f2019' if is
_colab else "."   #change dir to your project folder
# print("root_path now set to {}".format(root_path))
```

3.1 Data Sources

With v0 of EveryQualia, we downloaded the May 2012 Erowid Archive from `http://de1.erowid.org/erowid_archives/`, which is the most recent archive as of Dec 9, 2019. This download contains the entire mirror of the Erowid website from May 1st, 2012, and contains 19924 experience reports, each as its own html document. `@Yevbar` maintains a mirror of all the trip report raw files on his github, at Old Erowid Scraper.

One of the earliest and biggest unknowns for this thesis was whether there was going to be enough data on which to perform any meaningful analysis, since machine learning algorithms typically require a sizable amount of data. Early conversations with Lyle Ungar and Chris Callison-Burch suggested that a corpus of 200-300K reports was desirable. Since psychedelics are still, as of Dec 9, 2019, a Schedule 1 controlled substance, reports of psychedelic expereinces are not very easy to find, as every published account after the early 1970's is tantamount to admitting a federal crime. As such, a considerable amount of effort that is not necessarily documented here was really looking for sources for psychedelic trip reports. Feelers to the Johns Hopkins team revealed that the session reports in their clinical trials are private and not available to use for this thesis, and conversations with friends in the industry regarding an upcoming Johns Hopkins global survey study that would be the largest of its kind in the world revealed that it was still in IRB stages. Conversations with researchers at University of Oxford, who are also doing fascinating work with natural language processing on transcendent experiences accounts, revealved that their studies are based on in person interviews and their transcriptions, which the researchers personally and painstakingly obtained. Further details about this study cannot be shared in this thesis as their work serves as the basis of an upcoming PhD thesis, the details of which is not yet public as of this writing. In parallel Penn Society for Psychedelic Science, Harvard, Princeton (and 14 other schools, as of Dec 9, 2019) are growing the Intercollegiate Psychedelics Network (IPN). Multiple conversations were had with other student leaders about setting up

an Intercollegiate Psychedelics Archive, or IPN Archives, containing not only our respective student club files but also serve as a central source for aggregating trip reports from students. Indeed, there were some initial efforts of making this happen, but due to legal considerations, we decideed to hold off for now until the legal climate was more amenable, for our personal safety and the wellbeing of our respective student organizations.

And such, efforts were directed to other online sources, such as various reddit forum, and informal messaging boards, but none of which contained enough data at high enough quantities and structure as Erowid. David Yaden, a PhD candidate with Martin Seligman and Lyle Ungar, is doing fantastic work in assembling The Varieties Corpus, which is not yet available to the public and researchers. A natural extension of this thesis is to incorporate David's data in future work.

As my conversations with Yev progressed, he decided to try once again scraping Erowid live. Note that Erowid has pretty strict guidelines for crawlers, including:

- Run your spider only during off-peak hours (between midnight and 5am PST)
- Limit your robot to 10 file hits per second during off-peak hours and 2 file hits per second during peak hours.
- Robots must properly identify themselves. Archiving and spidering robots that do not identify themselves or that falsify their agent name will result in your IP(s) being blocked. Please set up your spider to do very short runs (less than 100 pages) and test it while you monitor the robot before simply launching it against our site and walking away.

Running a crawler at full speed (10 pages / second) non stop, crawling all 35,000 trip reports currently live (as of Dec 9, 2019), would take a full hour. This doesn't seem like a lot, but in reality the crawling limits are much lower, including a 1000 pages / day limit, and in practice we got repeatedly blocked by Erowid and it often took several hours to crawl 1000 pages. We entertained the idea of spinning up multiple AWS EC2 instances and using a distributed architecture to crawl Erowid (Yev actually wrote the code, and the idea was that each crawler would have a different IP address to "fool" Erowid). But this would be dishonest, and having not received an email reply from Erowid plus a variety of other reasons, I decided to not to proceed with scraping the entirety of Erowid. Indeed, this is an extension for further research.

Some important links about using the Erowid Experience Vaults:

- Erowid Spider Guidelines
- About the Erowid Experience Vault (`archive-20191208`)
- Erowid Experience Vault Review Process (`archive-2019-1208`)
- Erowid Downloadable Archives Collection

3.2 Data Scraping Procedure

In the end, we decided to stick with the corpus of EveryQualia v0, which was the May 1, 2012 archive of the full Erowid website from `http://de1.erowid.org/erowid_archives/`. To extract the data into a usable format, we unarchived the disk image, mounted it to disk, and navigated to the folder containing 19934 html files (see here). The following script was used to extract the report body, report title, and report substance(s) into a csv file:

```
import csv
from lxml import html
from os import listdir
from os.path import isfile, join

def trip_reports_to_csv():
    trip_csv = "trips.csv"
    columns = ["report", "title", "substance"]
    mypath = "experiences"
    onlyfiles = [join(mypath, f) for f in listdir(mypath) if isfile(join(mypath, f))]

    with open(trip_csv, "w") as trip_csv:
        trip_writer = csv.writer(trip_csv, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)
        trip_writer.writerow(columns)
        for f in onlyfiles:
            with open(f, "r", encoding='utf-8', errors='ignore') as f_:
                try:
                    page = f_.read()
                    tree = html.fromstring(page)
                    report = ''.join(tree.xpath('//div[@class="report-text-
surround"]/text()')).strip()
                    title = tree.xpath('//div[@class="title"]/text()')[0]
                    substance = tree.xpath('//div[@class="substance"]/text()')[0]
                    trip_writer.writerow([report, title, substance])
```

```
                except:
                    pass

trip_reports_to_csv()
```

The package lxml.html was used to match xpaths, with credits to Yev Barkalov for the xpaths that correctly matched the proper div elements for desired sections. In all, the script took about 1 hour to run on our hardware (rough estimate). It should be noted that there was certain demographics information such as gender, age, and publication date that were available (See Section 2.4, Previous Work), but were not included in the analysis of this thesis. This certainly hold promise for future research.

3.3 The Data: Raw and Unfiltered

In total, there were 19924 substance reports, each having a `report` body, a `title`, and a `substance` field.

In [13]:

```
import pandas as pd

trips_raw_filepath = '{}/trips.csv'.format(root_path)
pd_tripReports = pd.read_csv(trips_raw_filepath)
pd.set_option('display.width', 80)
# pd_tripReports.shape
# pd_tripReports.info()
```

In [14]:

```
# First 6 entries of the data
pd_tripReports.head()
```

Out[14]:

	report	title	substance
0	After having had some success with other forms...	Sideways World	Salvia divinorum (5x extract)
1	Me and a couple of my buddies decided one nigh...	Physical Wellbeing = Crucial	Mushrooms
2	My girlfriend and I had been saving the methyl...	The Artful Dodger	Methylone
3	I just want to warn anybody taking Lithium (or...	Seizure Inducing Combo	LSD & Lithium
4	I have had several attempts before this to bre...	Enlightenment Through a Chemical Catalyst	5-MeO-DMT

In [15]:

```
# Last 6 entries of the data
pd_tripReports.tail()
```

Out[15]:

	report	title	substance
19919	First off - I want to answer the question, 'Do...	My Truth About Seeds...	Morning Glory
19920	It was actually my mom who got me on Ritalin. ...	The Help I Hate to Love	Methylphenidate (Ritalin)
19921	This being a hopefully useful report on low-do...	Meeting the Spirit Plus a Territorial Dog	Peruvian torch cacti
19922	This journal shall hopefully be a guide for me...	A Journey Journal	Salvia divinorum (5x extract)
19923	I ingested 12-15mg of 2ce orally.\n\n\n\nI dum...	A Late Winter's Trip	2C-E

In [16]:

```
# An example trip report
report_example = pd_tripReports.loc[9000]
report_example["title"]
```

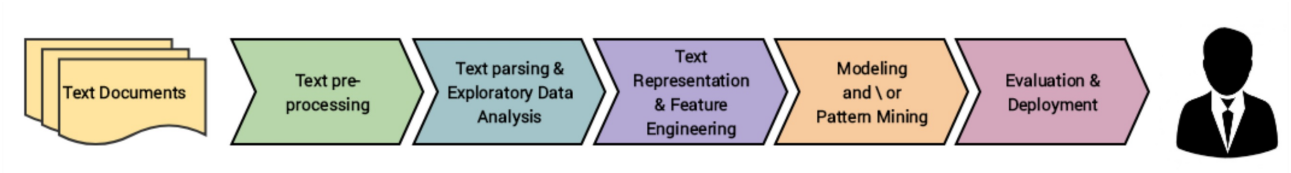
```
print("Title: {}".format(report_example["title"]))
print("Substance: {}".format(report_example["substance"]))
print("Report Body: {}".format(report_example["report"]))
```

```
Title: Hubris Nightmare
Substance: Mushrooms - P. cubensis
Report Body: I (or someone like me) had this bad trip in 1999. It taught me a lot about myself and
the mushroom.  I had the good/mixed fortune of coming into posession of a lot of dried/cultivated
P. cubensis. At the time, I had been through a lot in my personal life, including a divorce after
a decade of marriage--also financial crisis and crippling depression. I had a decent amount of fam
iliarity with many different psychoactive drugs including lots of trips with LSD and lower dose mu
shrooms (2-7 gms). I was going through a mental mix of self-loathing/'who-gives-a-shit-about-
anything' and some bizarre/dangerous 'psychedelic-machismo' (meaning, thinking I was some type of
superfreak capable of massive doses). I had never previously had a 'bad trip' on any substance.  I
was lonely and feeling very sorry for myself and considering that maybe I really was unworthy of l
ove and deserved to be miserable.  Obviously, this is a terrible set to go into a massive dose of
P. cubensis.  But wait!  It gets worse...add a terrible setting--that of a public rock concert ami
dst about 18000 people. I went alone.  I had one good thought--I drove, but brought taxi fare just
in case I was too fucked up to drive home. Thank God!



I had not initially planned on eating 10grams of the 'shrooms.  I ate 4 or so grams at the
beginning of the day-long show. After a while I was pleasantly tripping.  In the midst of my
longing to feel better, I began to 'nibble' at the large bag of shrooms.  Later I would realize I
had reached the approximate 10gram level. I was not worried.

Things began to turn sour...first, I became extremely self-conscious. I felt that people around me
were noticing my behaviour (though I was doing nothing unusual). The main (trippy-jam) band was no
w playing and there were likely many hundreds of people tripping.  Then, I began to 'hear the
chorus' of voices (clearly in my head) discussing me.  This is akin to the critical voices
reported by many people diagnosed with schizophrenia. I did not/do not have that illness and knew
it was an effect of the mushrooms. Yet, it was very unpleasant. I assumed it was the mushroom 'spe
aking' a la  Mazatecs/Terence McKenna etc., but I did not enjoy the 'message'!  Basically it/they
knew all about me--they (because the voices were distinct/plural) knew my every weakness, every ac
hilles heel and straightforwardly gored me with criticism after criticism, attack after attack.
They clearly despised me and hated me! They told me I was a phony, a 'poser', someone who thought
they were 'cool' but wasn't.  I began to feel panic and to generalize it to people around me--they
now had scared and angry looks in their eyes. Some of them were obviously drunk and starting to ge
t mean/belligerant, as drunks often do.



At that point the mushroom accelerated me--what I mean is, in an outdoor theater of 18000 packed p
artiers previously jamming to their favorite band, suddenly I (I still was I) was the only person
moving.  Everyone else had slowed to a very gradual slow-motion weave or had stopped moving-frozen
in time.  It was the freakiest thing I had ever seen.  Still, I could not enjoy this utterly
bizarre event.  I knew it was impossible, yet it was patently 'real' and continued with no break.
I looked down at the band. They seemed like they were a mile down the hill (my seats were bad, but
not that bad).  They would play one note--a lone sound from a guitar or keyboard and then stand
there. After what seemed like ten minutes later, another lone note would emanate. They stood
motionless and let their arms and heads hang down toward the ground like the robot on 'Lost in Spa
ce' when someone 'unplugged him'. I could not fucking believe this!  Meanwhile, as there was no
music to focus on (just a disconnected note seemingly every few minutes) all I could hear were the
attacking 'voices'. It was horrible!



I began to wonder if the people around me could even see me. Was I actually accelerated in time, a
s it most completely appeared? It was like that episode of Star Trek when Kirk got accelerated by
some weird aliens and the rest of the crew stood there for hours as he walked right by them! This
sounds amusing and could make for perhaps interesting adventures, but it wasn't! I began to think
if this was possible (and actually occurring), what the fuck might happen next?! I stood up and re
alized 'I've got to get out of here!'



The mocking sarcastic nasties kept after me--laughing and commenting on my flight. They would not
let up on me!  I had heard a 'voice' in past trips, but not a rogues gallery of assholes intent up
on my crucifiction like this.  I got to the parking lot and of course, since the show was not over
, there were thousands of cars.  I began to look where I thought my car was. I got more and more c
onfused and anxious. Then I remembered 'I can't drive when I am the only one moving!' Other cars m
ight be stopped in the freeway like obstacles.  I was again aware that I had taken mushrooms and w
as very, very tripadelic. Then I remembered the taxi fare--I ran to a main road, flagged down a ta
xi and went home. I was so grateful to be back in my apartment (which I hated and associated with
all my losses).
```

```
I was still tripping severely. I took 3mg of lorazepam and finally started to come down. I pieced
together this history, gauging from the amount of mushrooms left how much I had eaten. I felt like
a dumbfuck and the mushroom showed me that I clearly was one.  It was a harsh and painful lesson i
n humility.  It taught me that it was a very powerful force, not to be screwed with or forsaken.
After this experience I did not take mushrooms for several months.  Since then, when I have, I hav
e planned to be in a good and safe mindframe (set)--never to take them when feeling sad, depressed
or 'not right' in any way. I have come to believe that there is little room for 'recreation use' a
t all for me. It is instructive/soul-cleansing/spiritual, not 'recreational' and not-necessarily '
fun.' Mushrooms are a serious conscious endeavor and they deserve and will demand respect. In gene
ral, a sacred mindframe combined with meditation, in a safe/attractive place, not in public, predi
cts a good/beneficial trip. Please heed this advice from one who 'had to learn the hard way' when
taking any entheogenic trip of your chosing.

Peace and love
```

4. Data Cleaning and Exploratory Data Analysis (EDA)

4.1 Machine Learning, and the Natural Language Processing Pipeline



A high-level standard workflow for any NLP project

`Infographic Source`

Now that we have a collection of documents, i.e.: transcendent experience reports, obtained from the Erowid Experience Vaults, we can continue with the remainder of our analysis. Nautural Language Processing (NLP) is a subfield of machine learning, and deals primarily with unstructured textual data. The above graphic outlines the high level standard workflow for an NLP project, which includes:

- `Text Pre-processing`: since the web is messy and unpredictable, we need to clean our data appropriately and perform some pre-processing to prepare our data for subsequent analysis
- `Text Parsing & Exploratory Data Analysis`: what structures exist in the text, such as parts of speech or named entities (e.g.: `TOM` can be an organization (`ORG`) or a person (`PER`) depending on context) - text parsing helps us construct better understand underlying structures of the text; exploratory data analysis (EDA) is the process of playing and understanding the data. It's as if the data is a person, and you're asking it questions over coffee to get to know them better. No particular way of conversation is necessarily better than another, and data scientists can rely on tried and true techniques or get creative.
- `Text Representation & Feature Engineering`: The chosen *representation* of the data, as well as the features extracted from the raw data, are extremely important. Just as a landscape artist needs to choose the angle and form of the mountain he is painting, selecting the scenes and objects (features) to include to represent spring time, a data scientist needs

to choose the method(s) of representing the text, and select features to highlight. More frequent than not, a "feature" in natural language processing is a vector of integeters or real numbers (e.g.: counts of words per document).
- `Modeling and / or Pattern Mining`: depending on the task at hand, and the data (i.e.: metadata, labels etc.) available to us, we can use either supervised or unsupervised techniques to analyze our data. These techniques include but are not limited to topic modelling techniques, clustering (to find similar groups), and dimensionality reduction techniques (to visualize the data). A substantial portion of the thesis examines the upon the above techniques, with particular emphasis on topic modelling.
- `Evaluation & Deployment`: Once we have a model, we need to have a way to understand how well it performs. That is the topic of Evaluation, a crucial part of the natural language processing and machine learning pipeline. Without sound evaluation metrics, we have created a model, not a *good* model. Indeed, unsupervised learning techniques are typically hard to evaluate, and note that improving evaluation metrics is a further area of research.

Note that the workflow above has been drawn linearly, but the NLP pipeline in reality has many loops — extracting features, finding patterns, and asking questions about the data do not occur in a vaccuum. We present our work below linearly for structure, but the process of getting to know the data and writing this thesis was far from linear.

4.2 Text Preprocessing

4.2.1 Overview of Text Preprocessing

Text Preprocessing refers to a broad class of techniques that pre-processes the text before any analysis is done. It's analogous to a rough sieve filtering out the largest rocks, cleaning and normalizing the corpus on a high level. Note that there is no "canonical process" to perform text pre-processing. In practice, the types of preprocessing done depends on the type of data, the quantity of data, trial and error. Steps marked with `*` seem to be those that are most frequently used.

- `* Remove extra spaces`
- `* Make all text lowercase`
- `* Remove stopwords`
- `* Tokenization`
- `* Normalize accented characters`
- `* Remove special characters`
- `Remove rougue` html / xml `tags`
- `Remove numbers`
- `Expanding contractions`
- `Stemming`
- `Lemmatization`

For those interested, here is some more detail on each step:

- `* Remove extra spaces`:

  > a whitespace is not just a whitespace in NLP; they can be `space` or `\tab`, and depending on the parse, one needs to deal with newline characters such as `\n` or `\r`. Stripping away unecessary whitespace to find word boundaries is a crucial step in pre-processing. This is most commonly done with `str.strip()`

- `* Make all text lowercase`:

  > should `wow` be treated the same as `WOW`? In sentiment analysis tasks, capitalizations matter very much, as the presence of caps often connote delight. In other tasks that are more informational based, capitalization add noise and unwanted dimensionality. This is most commonly done with `str.lower()`

- `* Remove stopwords`:

  > words such as `a` and `the` and `and` and `is` occur so frequently in the english language that they often don't add any meaningful information to the task at hand, so we pre-empt noise by adding them to a `stopwords` list and remove them from our corpus. Care must be taken with stopwords as by adding a word to a stopwords list we are a priori claiming that word does not add value. Indeed, men tend to use `the` more than women, so a classifier that predicts gender would be well served to include the presence or counts of `the` as a feature (Lyle Ungar).

- `* Tokenization`:

  > the process of dividing a big chunk of text into individual units is called tokenization. In English, this is fairly

> straightforward: use individual words as tokens, or individual characters if we're building a character-level model. We also have to consider whether to make puntuations tokens, and whether we include any ngrams, e.g.: a bigram such as `cat jumped` or trigram such as `you are beautiful` separately as tokens, in addition to the individual words `cat`, `jumped`, `beautiful` etc. Tokenization is considerably harder for languages that don't have spaces, such as Chinese. In practice, we use built in tokenizers such as `nltk` or `spaCy`

- `* Normalize accented characters`:

> e.g.: `á` to `a`. Useful for english, but for other languages accented characters may have special meaning and usage. This is most commonly done with `unicodedata.normalize` and built in `sklearn` functionality

- `Remove rougue html / xml tags`:

> websites are written in `html` (`xml`), `css`, and `js`; If a website is a human being, `html` is the skeleton and meat, `css` is the makeup, and `js` provides the instructions for interactivity. As we know, sometimes bones break, but the human is still functioning — likewise, an `html` or `xml` file might be slightly corrupted, but the browser will still render the webpage. After we've scraped the webpage, we remove those `html` tags that are extraneous, so all we are left is the meat. This is most commonly done with packages such as `Beautiful Soup`

- `Remove numbers`:

> if we were designing a system to answer questions like `what is the population of norway`, numbers in the original data set is crucial. Other times, number add only noise. Depending on the problem at hand, the data scientist makes an informed decision of whether to remove numbers This is most commonly done with `regex`, or regular expressions, highly structured patterns that "match" to patterns of text. For example, the `regex` that matches to a 1 or more numbers is `[0-9]+`

- `Expanding contractions`:

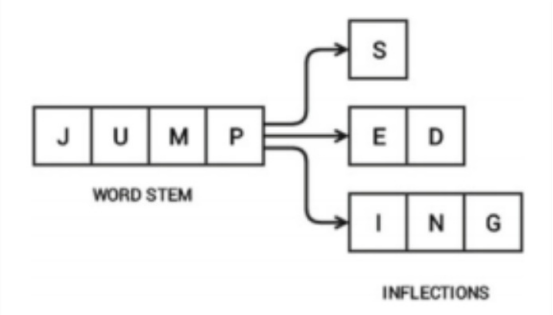
> e.g.: `don't` to `do not`. Once again, this is optional and depends on the task and availability of data. Lyle Ungar remarks this has been highly optional in his work.

- `* Remove special characters`:

> do special characters such as `<`, `!`, `#` add value? Indeed, sentiment analysis algorithms often look for `!`, and if we were analyzing twitter data, `#` are important. In other cases, special characters are treated as gibberish and removed.

- `Stemming`:

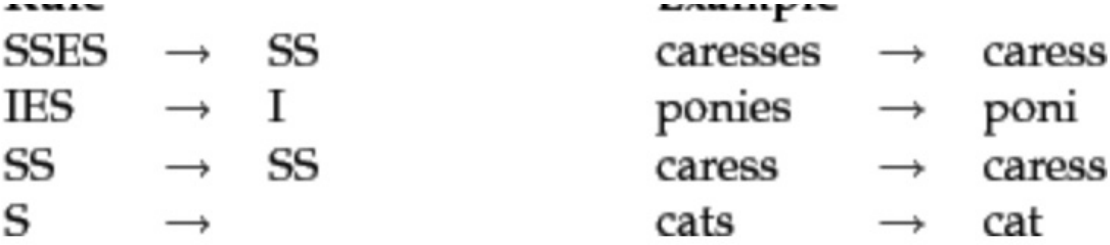
> `Stemming` refers to the often crude procedure of removing the ends of words to find the "word stem"; e.g.: walks -> walk; `Porter Stemmer`, `Lovins Stemmer`, `Paice Stemmer` are three examples of commonly used stemming algorithm. Some `Porter Stemmer` rules are found below (source)



Word stem and its inflections (Source: Text Analytics with Python, Apress/Springer 2016)

`Stemming Graphics Source` Here's an example of Porter Stemming rules, from the Stanford IR book (Manning, Raghavan, Hinrich Schütze, & University Of Cambridge, 2009)

Rule	Example



- `Lemmatization`:

  `Lemmatization`, to quote verbatim Stanford's Information Retrival Texbook (Manning, Raghavan, Hinrich Schütze, & University Of Cambridge, 2009), "usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma". In other words, lemmatization returns the "cannonical" form of a word. Some examples of lemmatization: (source)
  - rocks : rock
  - corpora : corpus
  - better : good; In a very real sense, lemmatization is a more complex procedure and can often produce more desirable results
    However, it is interesting to note that with sufficiently large datasets in industrial machine learning applications (such as those done at Google), stemming and lemmatization are uncommonly used (**source**: in conversation with Lyle Ungar)

- Finally, save cleaned text to disk: e.g.: `news_df.to_csv('news.csv', index=False, encoding='utf-8')`, OR use pickle dump, for ease of retrieval and use later

Note: Text preprossing procedures draws inspiration from here and A Practitioner's Guide to Natural Language Processing (Part I) — Processing & Understanding Text, as well as Professor `Chris Callison-Burch's` Computational Liguistics Class.

4.2.2 The Details of Text Pre-processing

In [72]:

```
# Heavily adapted from: https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72
### Getting the right packages

# for Colab only; OR, after installing SpaCy need to download the language models
# !pip install spacy
# !python -m spacy download en_core_web_sm
# !python -m spacy download en_core_web_md
# !python -m spacy download en_core_web_lg

import pandas as pd
import spacy
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize.toktok import ToktokTokenizer
import re
from bs4 import BeautifulSoup
import unicodedata
# From Spacy: English multi-task CNN trained on OntoNotes, with GloVe vectors trained on Common Crawl. Assigns word vectors, context-specific token vectors, POS tags, dependency parse and named entities.
# nlp_spacy = spacy.load('en_core_web_md', parse=True, tag=True, entity=True)
tokenizer = ToktokTokenizer()

print("Section 4.2.2 Text Pre-processing: done loading packages")

##### Recall the list of preprocessing tasks
# - `* Remove extra spaces`
# - `* Make all text lowercase`
# - `* Remove stopwords`
# - `* Tokenization`
# - `* Normalize accented characters`
# - `Remove rougue `html`/`xml` tags`
# - `Remove numbers`
# - `Expanding contractions`
```

```
# - `Stemming`
# - `Lemmatization`
```

```
Section 4.2.2 Text Pre-processing: done loading packages
```

In [17]:

```
# Option to reimport the data
pd_tripReports = pd.read_csv(trips_raw_filepath)
```

In [18]:

```
# make text lowercase
pd_tripReports["report"] = pd_tripReports["report"].str.lower()
pd_tripReports["title"] = pd_tripReports["title"].str.lower()
pd_tripReports["substance"] = pd_tripReports["substance"].str.lower()

pd_tripReports.head()
```

Out[18]:

	report	title	substance
0	after having had some success with other forms...	sideways world	salvia divinorum (5x extract)
1	me and a couple of my buddies decided one nigh...	physical wellbeing = crucial	mushrooms
2	my girlfriend and i had been saving the methyl...	the artful dodger	methylone
3	i just want to warn anybody taking lithium (or...	seizure inducing combo	lsd & lithium
4	i have had several attempts before this to bre...	enlightenment through a chemical catalyst	5-meo-dmt

In [21]:

```
# Prepare stopwords
# Note here we only use the stopwords from the nltk package;
# see section 4.2.3 for a more in depth treament of stopword removal
import nltk

nltk.download('stopwords')
stopword_list = nltk.corpus.stopwords.words('english')
stopword_list.remove('no')
stopword_list.remove('not')

# Remove stopwords
def remove_stopwords(text, stopword_list, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text

# remove_stopwords("The, and, if are stopwords, computer is not", stopword_list) # example
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/alextzhao/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [22]:

```
# * Normalize accented characters
def remove_accented_chars(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text

# remove_accented_chars('Sómě Áccěntěd těxt') # example
```

In [23]:

```
# Remove rougue `html`/`xml` tags
def strip_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text()
    return stripped_text

# strip_html_tags('<html><h2>Some important text</h2></html>') # example
```

In [24]:

```
# Remove Special Characters
def remove_special_characters(text, remove_digits=False):
    pattern = r'[^a-zA-z0-9\s]' if not remove_digits else r'[^a-zA-z\s]'
    text = re.sub(pattern, '', text)
    return text

# remove_special_characters("Well this was fun! What do you think? 123#@!",
#                           remove_digits=True) # example
```

In [25]:

```
# Simple Stemmer based on Porter Stemmer
# Lyle Ungar recommends not using a stemmer
def simple_stemmer(text):
    ps = nltk.porter.PorterStemmer()
    text = ' '.join([ps.stem(word) for word in text.split()])
    return text

# simple_stemmer("My system keeps crashing! his crashed yesterday, ours crashes daily") # example
```

In [26]:

```
# Lemmatize Text
# Lyle Ungar recommends not using a lemmatizer
def lemmatize_text(text):
    text = nlp(text)
    text = ' '.join([word.lemma_ if word.lemma_ != '-PRON-' else word.text for word in text])
    return text

# lemmatize_text("My system keeps crashing! his crashed yesterday, ours crashes daily") # example
```

In [27]:

```
# Adapted from https://towardsdatascience.com/a-practitioners-guide-to-natural-language-
processing-part-i-processing-understanding-text-9f4abfd13e72
def normalize_corpus(corpus, html_stripping=True,
                     accented_char_removal=True, text_lower_case=True,
                     special_char_removal=False, text_lemmatization=False,
                     text_stemming = False, stopword_removal=True,
                     remove_digits=True, contraction_expansion = False):
    """
    Input: corpus, list of documents
    Output: normalized_corpus, list of normalized documents that have been preprocessed
    """
    normalized_corpus = []
    # normalize each document in the corpus
    for doc in corpus:
        # strip HTML
        if html_stripping:
            doc = strip_html_tags(doc)
        # remove accented characters
        if accented_char_removal:
            doc = remove_accented_chars(doc)
        # expand contractions
        if contraction_expansion:
            doc = expand_contractions(doc)
        # lowercase the text
        if text_lower_case:
            doc = doc.lower()
        # remove extra newlines
```

```
        doc = re.sub(r'[\r|\n|\r\n]+', ' ',doc)
        # stem text: caution when stemming + lemmatizing text at the same time
        if text_stemming:
            doc = simple_stemmer(text)
        # lemmatize text
        if text_lemmatization:
            doc = lemmatize_text(doc)
        # remove special characters and\or digits
        if special_char_removal:
            # insert spaces between special characters to isolate them
            special_char_pattern = re.compile(r'([{.(-)!}])')
            doc = special_char_pattern.sub(" \\1 ", doc)
            doc = remove_special_characters(doc, remove_digits=remove_digits)
        # remove extra whitespace
        doc = re.sub(' +', ' ', doc)
        # remove stopwords
        if stopword_removal:
            doc = remove_stopwords(doc, stopword_list, is_lower_case=text_lower_case)

        normalized_corpus.append(doc)

    return normalized_corpus
```

In [29]:

```
## Preprocess our corpus
## Long run time: ~ 2 min

# pd_tripReports.info()
# pd_tripReports.dtypes
pd_tripReports["report"] = pd_tripReports["report"].astype('str')
pd_tripReports["title"] = pd_tripReports["title"].astype('str')
pd_tripReports["substance"] = pd_tripReports["substance"].astype('str')

corpus_list = pd_tripReports["report"].to_list()
corpus_list_normalized = normalize_corpus(corpus_list)

pd_tripReports_normalized = pd_tripReports
pd_tripReports_normalized["report"] = pd.Series(corpus_list_normalized)
```

In [31]:

```
# example trip report not preprocessed, first 400 chars
pd_tripReports.loc[19922, "report"][:400]
```

Out[31]:

```
'this journal shall hopefully be a guide for me into better understanding the effects of salvia di
vinorum. like many journals i have attempted to write, they all end in lack of desire to finish th
em. i plan to make this journal different. it is my hope that i describe each experience in the fu
llest extent to better understand the unique nature of this plant. \n\n\n\n9/19/2002\n\n\n\ni rece
ived two grams of'
```

In [35]:

```
# example trip report preprocessed, first 400 chars
pd_tripReports_normalized = pd.read_csv("trips_normed-html-accent-special-digits.csv")
pd_tripReports_normalized.loc[19922, "report"][:400]
```

Out[35]:

```
'journal shall hopefully guide better understanding effects salvia divinorum. like many journals a
ttempted write , end lack desire finish them. plan make journal different. hope describe
experience fullest extent better understand unique nature plant. 9/19/2002 received two grams 5x e
xtract. excited first experience called friend watch , since first time not know would react.
prepared water pipe pu'
```

In [34]:

```
# OPTIONAL
# save preprocessed file to disk
# pd_tripReports.to_csv('{}/trips_normed-html-accent-special-digits'.format(root_path),
index=False, encoding='utf-8')
```

Because the pre-processing, especially the removal of stopwords takes a substantial amount of time to run, we store a serialized version of the object so we may retrive it later at a substantially faster rate; see pickle tutorial

In [0]:

```
## OPTIONAL
# Use this cell as a template to save and load large data files;
# replace filenames appropriately;
# Pandas data frames can be saved directly to csv files with `pd_df.to_csv("filename")`


# import pickle

# ### Stores the tokenized reports:
# with open('', 'wb') as out_file:
#     pickle.dump(reports_text_tokenized, out_file)

# ### Loads the tokenized reports:
# with open('reports_tokenized_map', 'rb') as in_file:
#     reports_text_tokenized = pickle.load(in_file)
```

4.2.3 A Note on Preparing Stopwords

The removal of stopwords (typically commonly occuring, low information content words) in natural language processing is in itself an art. Both automatic detection of stopwords (Wilbur & Sirotkin, 1992) and dictionary approaches have been widely used. For example, `nltk` has 179 stopwords , `wordcloud` package has 192 stopwords, and the `sklearn` package has 318 stopwords, with 116 words shared between all three lists. Some examples of these shared stopwords are 'how', 'itself', 'an', 'ours', 'who', 'had', 'once', 'before', 'yours', and 'the', which are among the most common words in the English language.

In the previous text pre-processing step, we simply used the 179 stopwords from `nltk` . Here, we prepare an extended list of stopwords, to be used selectively downstream in topic modelling (Section 6) and word cloud generation (Section 7). The extended stopwords list is prepared by (1) using `nltk` and `wordcloud` stopwords as a baseline (exclude `sklearn` stopwords because of known issues and (2) adding all alternative spellings of drugs observed from EDA to our stopwords list; Note that the latter is an optional, concious design decision: we can leave the drug names be (in which case they would show up disproportionately in the word clouds), or we can remove all drug names from the reports and focus on non drug names that correlate with particular reports. In total, our psychedelic drug corpus has 2783 unique spellings / mispellings of drug names. In total, our extended stopword list consisits of 2999 unique stopwords.

In [434]:

```
# Retrieve all alternative spellings for substances
# Using R, we found out how many unique substance spellings there were, and stored in
`substances_unique` textfile
with open('substances_unique') as f:
    substances_all_alternative_spellings = f.readlines()
substances_all_alternative_spellings = [spelling.strip().replace("'", "") for spelling in substance
s_all_alternative_spellings]
substances_all_alternative_spellings[:30]

## prepare stopwords
stopword_list = []
stopwords_nltk = nltk_stopwords.words('english')
stopwords_wordcloud = list(wordcloud_STOPWORDS)
stopword_list.extend(stopwords_nltk + stopwords_wordcloud + substances_all_alternative_spellings)
# deduplicate list
stopword_list = list(dict.fromkeys(stopword_list))

# How many unique stopwords in total do we have?
len(stopword_list)
```

Out[434]:

```
2999
```

In [85]:

```
# Number of stopwords for each dictionary
from sklearn.feature_extraction import stop_words
len(stop_words.ENGLISH_STOP_WORDS) #sklearn
```

```
len(nltk_stopwords.words('english')) #nltk
len(wordcloud_STOPWORDS) #wordcloud
len(substances_all_alternative_spellings) #all substance spellings

# overlap between sklearn, nltk, and wordcloud stopwords
overlapped_stopwords = []
overlap_stopwords = [word for word in stop_words.ENGLISH_STOP_WORDS if word in
nltk_stopwords.words('english') and word in wordcloud_STOPWORDS]
print(overlap_stopwords[:20])
```

```
['each', 'further', 'at', 'nor', 'when', 'them', 'about', 'had', 'over', 'all', 'as', 'himself', '
same', 'on', 'can', 'it', 'under', 'yours', 'the', 'down']
```

In [0]:

```
# # NOTE: Long Run Time

# # Tokenize the text
# reports_text_tokenized = {}

# # reports_text is {substance: text} mapping
# # NOTE: This code takes an extremely long time to run
# for substance in reports_text:
#     substance_text = reports_text[substance]
#     tokens = nltk.word_tokenize(substance_text)
#     tokens = [token.strip().lower() for token in tokens]
#     substance_tokenized_text = ' '.join([token for token in tokens if token not in
stopword_list])

#     reports_text_tokenized[substance] = substance_tokenized_text
```

4.2.4 Conclusions from Text Pre-processing

Text Pre-processing serves as a rough filter that removes the most low information aspects of our corpus and provides preliminary normalization so we can begin better analyzing our data. Certain operations are very common and fairly non-destructive, such as making all text lower case, tokenization, removing special characters, and removing extra spaces. That being said, one size does not fit all. This is especially true for engineering the stopwords list, where we are deliberately throwing away data before we even look at it. This could be problematic, as certain NLP tasks such as predicting gender from text precisely relies on signals from common words such as "the" (e.g.: males empirically use "the" more often than females, source: Lyle Ungar), which is commonly thrown away in the text pre-processing stage. Tokenization can also be non trivial, with options for bigrams, trigrams, character level tokenization, and beyond. Stemming and Lemmatization can also affect downstream analysis drastically, and with sufficiently large datasets, they are often not performed to preserve information (source: Lyle Ungar). Ultimately, the steps involved in text pre-processing are up to the judgement of the data scientist. In this thesis, we have chosen to be fairly conservative in our text pre-processing, choosing preserving information over being too selective. This may add noise to subsequent analysis, and a natural extension is to examine more closely the effects of pre-processing on subsequent analysis.

4.3 Data Wrangling and Exploratory Data Analysis (EDA)

After preliminiary data preprocessing (cleaning), we perform data wrangling and Exploratory Data Analysis (EDA). Data Wrangling describes a suite of procedures to clean the data and transform the data into the right format for subsequent analysis, and is often the most time consuming stage of a data science pipeline. EDA refers broadly to procedures designed to better understand the data set through visualizations and summary statistics.

Here the EDA was mostly performed in R, in particular using `dplyr`, and was performed directly on the raw data. For reproducibility, the executible R code is also reproduced below; very similar procedures can be used with `pandas`.

In [37]:

```
# Sets up the ability to use R in a jupyter notebook
# https://stackoverflow.com/questions/39008069/r-and-python-in-one-jupyter-notebook
# To use R, add %%R to the beginning of a cell, before any code and any comments
%load_ext rpy2.ipython
```

In [39]:

```
%%R
# Installs the proper R packages
# Depending on local environment (e.g.: whether R packages for Anaconda are installed, this may or
```

```
# Depending on local environment (e.g.: whether R packages for Anaconda are installed, this may or
# may not be necessary)

# if(!require('pacman')) {
#   install.packages('pacman', repos = "http://cran.us.r-project.org")
# }
# pacman::p_load(dplyr, leaps, car, tidyverse, GGally, reshape2, data.table, ggcorrplot, bestglm,
glmnet, mapproj, pROC, data.tale, tm, SnowballC, wordcloud, RColorBrewer, reshape2)
```

```
NULL
```

In [40]:

```
%%R

## Read in the raw trip reports
tripReports <- fread("trips.csv")
tripReports.df <- as.data.frame(tripReports)
# glimpse(tripReports)

# Number of trip reports in our dataset: 19924
# nrow(tripReports)
```

In [41]:

```
%%R
# Examine the substances present in the dataset, writing to output file
tripReports %>% select(substance) %>% write.table(file = "substances-all", append=FALSE, col.names
= FALSE, row.names = FALSE, sep = "\n")

# # Examine the substances present in the dataset, display
tripReports %>% select(substance) %>% head(n=10)
```

```
                      substance
 1: Salvia divinorum (5x extract)
 2:                     Mushrooms
 3:                     Methylone
 4:                 LSD & Lithium
 5:                     5-MeO-DMT
 6: Salvia divinorum (6x extract)
 7:                      Cannabis
 8:                        2C-T-2
 9:                         6-APB
10:                       2-C-T-2
```

In [42]:

```
%%R

## Manipulate the dataframe to have 1 hot encoding for substances
tripReports$substance <- tolower(tripReports$substance)
tripReports <- as.data.frame(tripReports)

# Explicity define the substance of interest
substances.of.interest <- c("substance.mushrooms", "substance.lsd", "substance.mescaline",
"substance.cannabis", "substance.mdma", "substance.ayahuasca", "substance.nitrous_oxide",
"substance.salvia", "substance.methamphetamine", "substance.dmt", "substance.5_meo_dmt",
"substance.alcohol", "substance.ketamine", "substance.ibogaine", "substance.pcp", "substance.kava"
, "substance.kratom", "substance.morning_glory", "substance.syrian_rue", "substance.unknown",
"substance.UNK")

for (sub in substances.of.interest) {
  tripReports[sub] = 0
}

### Adding one hot encodings to the substances
tripReports$substance.mushrooms[grepl("mushroom|mushhrooms|mushooms|psilocin|psilocybin|psilocybe"
, tripReports$substance)] <- 1
tripReports$substance.lsd[grepl("lysergic acid|lsd", tripReports$substance)] <- 1
tripReports$substance.mescaline[grepl("mescaline|peyote", tripReports$substance)] <- 1
tripReports$substance.cannabis[grepl("cannabis|canabis|cannabbis|cannabinoid|cannabus|cannibis|canr
us| thc ", tripReports$substance)] <- 1
tripReports$substance.mdma[grepl("mdma|ecstacy|ecstasy|ectasy|molly", tripReports$substance)] <- 1
```

```
tripReports$substance.ayahuasca[grepl("ayahuasca|ayanausca|ayanusca|p.
viridis|p.viridis|b.caapi|b. caapi|cappi|viridis", tripReports$substance)] <- 1
tripReports$substance.nitrous_oxide[grepl("nitric|nitrites|nitrogen|nitrous|whippets", tripReports
$substance)] <- 1
tripReports$substance.salvia[grepl("salia|saliva|salivia|sally|salva|salvia|salvinorin",
tripReports$substance)] <- 1
tripReports$substance.methamphetamine[grepl("met|meth|methampetamine|methamphetamine|speed",
tripReports$substance)] <- 1
# dmt
tripReports$substance.dmt[grepl("nn-dmt", tripReports$substance)] <- 1
tripReports$substance.dmt[tripReports$substance == "dmt"] <- 1 # cannot just grep dmt otherwise th
is will include 5-meo-dmt
#5-meo-dmt
tripReports$substance.5_meo_dmt[grepl("5 meo-dmt|5-meo dmt|5-meo-dmt|5meo-dmt",
tripReports$substance)] <- 1
tripReports$substance.alcohol[grepl("alchohol|alcohol", tripReports$substance)] <- 1
tripReports$substance.ketamine[grepl("ketamin|ketamine", tripReports$substance)] <- 1
tripReports$substance.ibogaine[grepl("iboga|ibogaine", tripReports$substance)] <- 1
tripReports$substance.pcp[grepl("pcp", tripReports$substance)] <- 1
tripReports$substance.kava[grepl("kava", tripReports$substance)] <- 1
tripReports$substance.kratom[grepl("kratom", tripReports$substance)] <- 1
tripReports$substance.morning_glory[grepl("glory|glories", tripReports$substance)] <- 1
tripReports$substance.syrian_rue[grepl("syrian rue|rue", tripReports$substance)] <- 1
tripReports$substance.unknown[grepl("unknown|unidentified|unkown", tripReports$substance)] <- 1

glimpse(tripReports)
```

```
Observations: 19,924
Variables: 24
$ report                    <chr> "After having had some success with other f…
$ title                     <chr> "Sideways World", "Physical Wellbeing = Cru…
$ substance                 <chr> "salvia divinorum (5x extract)", "mushrooms…
$ substance.mushrooms       <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0…
$ substance.lsd             <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.mescaline       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.cannabis        <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0…
$ substance.mdma            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.ayahuasca       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.nitrous_oxide   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.salvia          <dbl> 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.methamphetamine <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0…
$ substance.dmt             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.5_meo_dmt       <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.alcohol         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.ketamine        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.ibogaine        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.pcp             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.kava            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.kratom          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0…
$ substance.morning_glory   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.syrian_rue      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.unknown         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.UNK             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
```

In [0]:

```
%%R

## Number of reports containing each substance: NOTE: a report might have more than one substance
tripSubstances <- tripReports %>% select(-report, -title, -substance)
tripReportsCount <- data.frame(substance = names(tripSubstances), count = colSums(tripSubstances))
tripReportsCountSorted <- tripReportsCount %>% arrange(desc(count))
tripReportsCountSorted
```

```
                   substance count
1          substance.cannabis  3110
2        substance.mushrooms  1686
3           substance.salvia  1556
4             substance.mdma  1188
5              substance.lsd  1131
6  substance.methamphetamine   929
7          substance.alcohol   928
8    substance.morning_glory   427
9    substance.nitrous_oxide   300
10       substance.5_meo_dmt   297
```

```
11      substance.syrian_rue   293
12        substance.ketamine   289
13          substance.kratom   207
14       substance.ayahuasca   172
15             substance.dmt   167
16            substance.kava   167
17             substance.pcp    81
18       substance.mescaline    73
19         substance.unknown    47
20        substance.ibogaine    43
21             substance.UNK     0
```

In [0]:

```
%%R

## Find reports with only one unique substance
# Note: Long run time ~ 40 seconds

tripReports$substance.unique_label <- "NA"
# glimpse(tripReports)
uniqueSubstanceRows <- tripReports %>% select(-report, -title, -substance, -substance.unique_label)
%>% rowSums() == 1

for (row in 1:nrow(tripReports)) {
  # this row contains a unique substance
  if (uniqueSubstanceRows[row]) {
    for (substance in substances.of.interest) {
      if (tripReports[row, substance] == 1) {
        tripReports[row, ]$substance.unique_label<- substance
      }
    }
  }
}

# glimpse(tripReports)
```

```
Observations: 19,924
Variables: 25
$ report                    <chr> "After having had some success with other f…
$ title                     <chr> "Sideways World", "Physical Wellbeing = Cru…
$ substance                 <chr> "salvia divinorum (5x extract)", "mushrooms…
$ substance.mushrooms       <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0…
$ substance.lsd             <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.mescaline       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.cannabis        <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0…
$ substance.mdma            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.ayahuasca       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.nitrous_oxide   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.salvia          <dbl> 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.methamphetamine <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0…
$ substance.dmt             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.5_meo_dmt       <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.alcohol         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.ketamine        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.ibogaine        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.pcp             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.kava            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.kratom          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0…
$ substance.morning_glory   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.syrian_rue      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.unknown         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.UNK             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ substance.unique_label    <chr> "substance.salvia", "substance.mushrooms", …
```

In [0]:

```
%%R

# TEMP, SAVE: Use this code to save the encoded table for future use in multiple formats
# save(tripReports, file="tripReportsEncoded.Rda")
# write.table(tripReports, file="tripReportsEncodedTable", sep = ";;", row.names = TRUE, col.names
= TRUE )
# write.csv(tripReports, file="tripReportsEncoded.csv", sep = ",", row.names = TRUE, col.names = T
RUE)
```

In [ ]:

```
%%R

#### Create a data frame that combines the count for reports containing a substance and reports th
at are uniquely a single substance
tripReportsUniqueSubstanceCountSorted <- tripReports %>% group_by(substance.unique_label) %>%
summarise(count = n()) %>% arrange(desc(count))
colnames(tripReportsUniqueSubstanceCountSorted) = c("substance", "count")
tripReportsUniqueSubstanceCountSorted %>% glimpse()

tripReportsCount_merged <- merge(tripReportsCountSorted, tripReportsUniqueSubstanceCountSorted, by
="substance")
colnames(tripReportsCount_merged) = c("substance", "n_includes_substance", "n_single_substance")
```

In [0]:

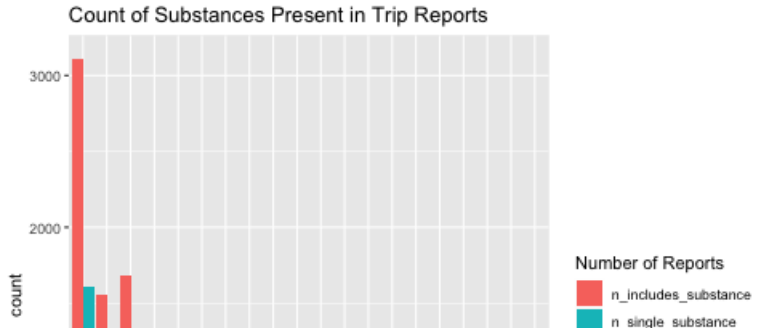
```
%%R
# EXPORT: Table of selected substances and their counts
tripReportsCount_merged %>% arrange(desc(n_includes_substance))
```

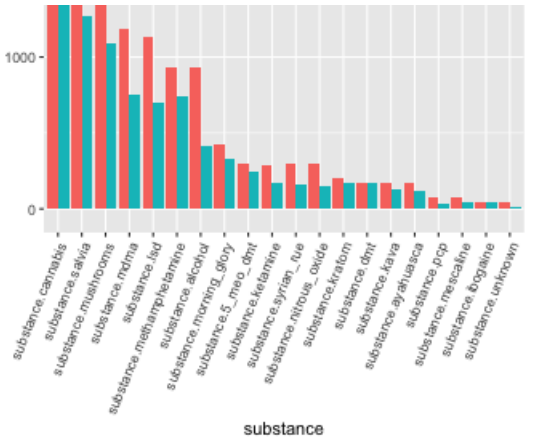
	substance	n_includes_substance	n_single_substance
1	substance.cannabis	3110	1609
2	substance.mushrooms	1686	1094
3	substance.salvia	1556	1270
4	substance.mdma	1188	747
5	substance.lsd	1131	696
6	substance.methamphetamine	929	746
7	substance.alcohol	928	409
8	substance.morning_glory	427	328
9	substance.nitrous_oxide	300	155
10	substance.5_meo_dmt	297	250
11	substance.syrian_rue	293	164
12	substance.ketamine	289	170
13	substance.kratom	207	171
14	substance.ayahuasca	172	114
15	substance.dmt	167	167
16	substance.kava	167	131
17	substance.pcp	81	38
18	substance.mescaline	73	44
19	substance.unknown	47	13
20	substance.ibogaine	43	39

In [0]:

```
%%R

## This data frame that encodes how many trip reports contains a substance and how many reports ar
e uniquely such a substance'
tripReportsCount_merged.long <- reshape2::melt(tripReportsCount_merged)
# EXPORT: Graph of distribution of substances
tripReportsCount_merged.long %>% ggplot( aes(x = reorder(substance, -value), y = value, fill=variab
le)) + geom_bar(stat="identity", position = "dodge") + theme(axis.text.x = element_text(angle = 67.
5, hjust = 1)) + ggtitle("Count of Substances Present in Trip Reports") + xlab("substance") + ylab(
"count") + labs(fill = "Number of Reports")
```

```
R[write to console]: Using substance as id variables
```





4.3.1 Treatment of `substance` column

The `substance` column is of type string, and describes what substance(s) were associated with a particular trip report. It turns out that users often co-consume multiple substances at once, and may convey that fact differently in different formats as plaintext. For example, supposed a user has had `lsd`, `mushrooms`, and `lithium` in a single session. They may record this fact as "`lsd, mushrooms, and lithium`", "`lsd, lithium & mushrooms`", "`mushrooms, lsd and lithium`" or in another way, using freely `,`, `and`, and `&` as delimiters. As such, great care had to be taken using `grep`, regular expressions, and multiple string operations to label each report with a collection of substances in a standardized format.

We also observed a tremendous number of unique substances in the `substance` column. Here we describe a few notable characteristics, and key procedures we performed:

- The were `2783` unique spellings for the `substance` column, accounting for mistakes, alternative spellings, and parenthetical clarifications (e.g.: `mda? (sold as ecstasy)`. A few ubstances such as `2-cb` had no alternative spellings, while `salvia` had 130 unique spellings! (a few examples taken verbatim include: `salvia`, `sally divinorum (5x extract)`, `salvia d`, `saliva`, `salvinorin-a`)
- Due to the overwhelming number of alternative spellings, and mistakes, manual deduplication was necessary using prior knowledge. After manual depupplication, we had `1379` unique substance labels. It is very possible that I have missed a considerable chunck of duplicated entries, and an interesting task in itself is to further analyze the substance names.
- The substance names were standardized with the format `substance.<normalized-name>`

With these considerations, we used one hot encoding to denote the presence of a substance, and also added a `substance.unique_label` column for denoting those reports that only feature one substance.

In [3]:

```
# Results of the encoding

import pandas as pd
root_path = "."
pd_trips_encoded_from_R = pd.read_csv('{}/tripReportsEncoded.csv'.format(root_path))
pd_trips_encoded_from_R.head()
```

Out[3]:

	Unnamed: 0	report	title	substance	substance.mushrooms	substance.lsd	substance.mescaline	substance.cannabis	sub
0	1	After having had some success with other forms...	Sideways World	salvia divinorum (5x extract)	0	0	0	0	
1	2	Me and a couple of my buddies decided one nigh...	Physical Wellbeing = Crucial	mushrooms	1	0	0	0	
		My girlfriend							

	Unnamed: 0	report	title	substance	substance.mushrooms	substance.lsd	substance.mescaline	substance.cannabis	sub
2	3	and I had been saving the methyl...	The Artful Dodger	methylone	0	0	0	0	
3	4	I just want to warn anybody taking Lithium (or...	Seizure Inducing Combo	lsd & lithium	0	1	0	0	
4	5	I have had several attempts before this to bre...	Enlightenment Through a Chemical Catalyst	5-meo-dmt	0	0	0	0	

5 rows × 26 columns

Since the EDA was performed on the raw data, here we normalize our corpus with our text pre-processing pipeline, and store the resulting data in `pd_trips_encoded_normalized` for later use

In [4]:

```
# CAUTION: Long Run Time
# EXPORT
# pre-process the encoded data frame to be the "master data frame" for subsequent analysis
pd_trips_encoded_from_R.dtypes
pd_trips_encoded_from_R["report"] = pd_trips_encoded_from_R["report"].astype('str')
pd_trips_encoded_from_R["title"] = pd_trips_encoded_from_R["title"].astype('str')
pd_trips_encoded_from_R["substance"] = pd_trips_encoded_from_R["substance"].astype('str')

corpus_list_encoded = pd_trips_encoded_from_R["report"].to_list()
corpus_list_encoded_normalized = normalize_corpus(corpus_list_encoded)

pd_trips_encoded_normalized = pd_trips_encoded_from_R
pd_trips_encoded_normalized["report"] = pd.Series(corpus_list_encoded_normalized)

# change column names to better fit pandas workflow
pd_trips_encoded_normalized.columns = pd_trips_encoded_normalized.columns.str.replace('.', '_')

# encoded and pre-processed data frame
# pd_trips_encoded_normalized.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-fccdffe2e122> in <module>
      8
      9 corpus_list_encoded = pd_trips_encoded_from_R["report"].to_list()
---> 10 corpus_list_encoded_normalized = normalize_corpus(corpus_list_encoded)
     11
     12 pd_trips_encoded_normalized = pd_trips_encoded_from_R

NameError: name 'normalize_corpus' is not defined
```

4.3.2 Facts about the Data: Trip Reports Length

- The mean report length is 542.5 words, or 3314.9 characters
- The median report length is 423 words, or 2576 characters
- The minimum report length is 4 words, or 18 characters; this report is `'passed drug test !'`
- The maximum report length is 8230 words, or 52707 characters

In [7]:

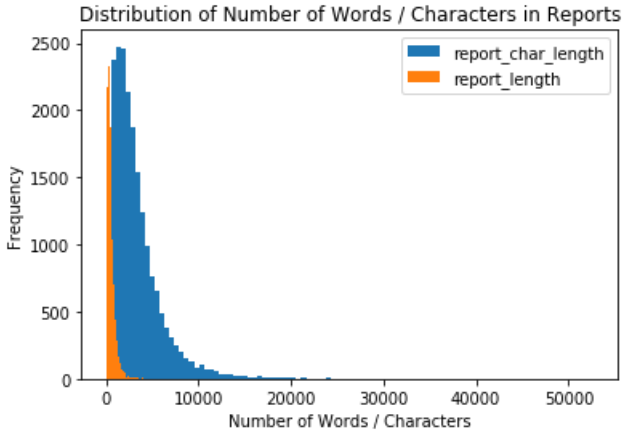
```
# README: Use the following import to load the cleaned data back into memory
# EXPORT:
import pandas as pd
pd_trips_encoded_normalized = pd.read_csv("pd_trips_encoded_normalized.csv")
```

In [438]:

```
import matplotlib.pyplot as plt
# average length of each report (character level)
pd_trips_encoded_normalized = pd_trips_encoded_normalized.assign(report_char_length =
pd_trips_encoded_normalized["report"].str.len())

# average lenght of each report (word level)
pd_trips_encoded_normalized = pd_trips_encoded_normalized.assign(report_length =
pd_trips_encoded_normalized["report"].str.split().str.len())

# plotting
fig = plt.figure()

ax_report_char_length = pd_trips_encoded_normalized["report_char_length"].plot.hist(bins = 100)
ax_report_char_length.legend()
ax_report_char_length.set_xlabel("Number of Characters")
ax_report_char_length.set_title("Number of Characters in Reports")

ax_report_length = pd_trips_encoded_normalized["report_length"].plot.hist(bins = 100)
ax_report_length.legend()
ax_report_length.set_xlabel("Number of Words / Characters")
ax_report_length.set_title("Distribution of Number of Words / Characters in Reports")

# mean and media of report lengths
# print("Mean report length is {}".format(pd_trips_encoded_normalized["report_length"].mean()))
# print("Mean report length (in chars)  is
{}".format(pd_trips_encoded_normalized["report_char_length"].mean()))

# print("Median report length is
{}".format(pd_trips_encoded_normalized["report_length"].median()))
# print("Median report length (in chars)  is
{}".format(pd_trips_encoded_normalized["report_char_length"].median()))

# print("Min report length is {}".format(pd_trips_encoded_normalized["report_length"].min()))
# print("Min report length (in chars)  is
{}".format(pd_trips_encoded_normalized["report_char_length"].min()))

# print("Max report length is {}".format(pd_trips_encoded_normalized["report_length"].max()))
# print("Max report length (in chars)  is
{}".format(pd_trips_encoded_normalized["report_char_length"].max()))

# obtaining the report with minimum length
# print('Shortest report was "
{}"'.format(pd_trips_encoded_normalized.iloc[pd_trips_encoded_normalized["report_char_length"].idxı
]["report"]))
```

Out[438]:

```
Text(0.5, 1.0, 'Distribution of Number of Words / Characters in Reports')
```



4.3.3 Facts about the Data: Substance Consumption and Co-consumption statistics

- 61.5\% of LSD users only consumed LSD, 29.6\% consumed LSD + another substance, and 8.9\% of users consumed 2 or more substances along with LSD; the most commonly co-consumed substances with LSD were cannabis (43.3\%), mdma

(19\%), alcohol (8.8\%), and Mushrooms (8\%)

- 64.9\% of Mushrooms users only consumed mushrooms, 27.1\% of users consumed mushrooms + another substance, and 8\% of of users consumed 2 or more substances along with mushrooms; the most commonly co-consumed substance with mushrooms were cannabis (56.8\%), alcohol (9.4 \%), MDMA (8.6\%), and LSD (6.4\%)
- 62.9\% of MDMA users only consumed MDMA, 24.2\% of users consumed MDMA + another substance, and 12.9\% of of users consumed 2 or more substances along with mushrooms; the most commonly co-consumed substances with MDMA were cannabis (33.2 \%), LSD (18\%), alcohol (13.4\%), and Mushrooms (10.1 \%)
- 60.3\% of Mescaline users only consumed mescaline, 24.7\% of users consumed mescaline + another substance, and 12.9\% of uses consumed 2 more more substances along with mescaline;the most commonly co-consumed substances with Mescaline were cannabis (47.1\%), LSD (13.7\%), mushrooms (13.7\%), and MDMA (7.8\%)
- 84.2\% of 5-MeO-DMT users only consumed 5-MeO-DMT, 12.9\% of users consumed 5-MeO-DMT + another substance, and 2.9\% of of users consumed 2 or more substances along with 5-MeO-DMT; the most commonly co-consumed substances with 5-MeO-DMT were cannabis (20.3\%), syrian rue (20.3\%), MDMA (15.3 \%), and salvia (11.9\%)
- 100% of DMT users only consumed DMT

In [57]:

```
%%R
# Perform subsequent analysis in R
R_trips_encoded_normalized = fread("pd_trips_encoded_normalized.csv")
R_trips_encoded_normalized <- as.data.frame(R_trips_encoded_normalized)
```

In [166]:

```
%%R
# Create Number of substance labels for each report
R_trips_encoded_normalized_eda <- R_trips_encoded_normalized %>%
    mutate(num_substance_labels = rowSums(select(., starts_with("substance_"),
-substance_unique_label)))
```

In [165]:

```
%%R
## MOST FREQUENT SUBSTANCES CO-CONSUMED WITH SELECTED SUBSTANCES


## Substances most frequently co-consumed with LSD
R_trips_encoded_normalized_eda %>% filter(substance_lsd == 1) %>%
    select(-substance_lsd) %>%
    gather(substance_key, isPresent, substance_mushrooms:substance_UNK) %>%
    filter(isPresent == 1) %>%
    group_by(substance_key) %>%
    summarise(count = n()) %>%
    mutate(percent = round(count / sum(count) * 100, digits = 1)) %>%
    arrange(desc(count))

# Substances most frequently co-consumed with Mushrooms
R_trips_encoded_normalized_eda %>% filter(substance_mushrooms == 1) %>%
    select(-substance_mushrooms) %>%
    gather(substance_key, isPresent, substance_lsd:substance_UNK) %>%
    filter(isPresent == 1) %>%
    group_by(substance_key) %>%
    summarise(count = n()) %>%
    mutate(percent = round(count / sum(count) * 100, digits = 1)) %>%
    arrange(desc(count))


# Substances most frequently co-consumed with MDMA
R_trips_encoded_normalized_eda %>% filter(substance_mdma == 1) %>%
    select(-substance_mdma) %>%
    gather(substance_key, isPresent, substance_mushrooms:substance_UNK) %>%
    filter(isPresent == 1) %>%
    group_by(substance_key) %>%
    summarise(count = n()) %>%
    mutate(percent = round(count / sum(count) * 100, digits = 1)) %>%
    arrange(desc(count))


# Substances most frequently co-consumed with mescaline
R_trips_encoded_normalized_eda %>% filter(substance_mescaline == 1) %>%
    select(-substance_mescaline) %>%
    gather(substance_key, isPresent, substance_mushrooms:substance_UNK) %>%
```

```
    filter(isPresent == 1) %>%
    group_by(substance_key) %>%
    summarise(count = n()) %>%
    mutate(percent = round(count / sum(count) * 100, digits = 1)) %>%
    arrange(desc(count))

# Substances most frequently co-consumed with dmt
# No co-consumption
R_trips_encoded_normalized_eda %>% filter(substance_dmt == 1) %>%
    select(-substance_dmt) %>%
    gather(substance_key, isPresent, substance_mushrooms:substance_UNK) %>%
    filter(isPresent == 1) %>%
    group_by(substance_key) %>%
    summarise(count = n()) %>%
    mutate(percent = round(count / sum(count) * 100, digits = 1)) %>%
    arrange(desc(count))


# Substances most frequently co-consumed with 5-MeO-dmt
R_trips_encoded_normalized_eda %>% filter(substance_5_meo_dmt== 1) %>%
    select(-substance_5_meo_dmt) %>%
    gather(substance_key, isPresent, substance_mushrooms:substance_UNK) %>%
    filter(isPresent == 1) %>%
    group_by(substance_key) %>%
    summarise(count = n()) %>%
    mutate(percent = round(count / sum(count) * 100, digits = 1)) %>%
    arrange(desc(count))

## ARCHIVED:
# R_trips_encoded_normalized %>% filter(substance_5_meo_dmt == 1) %>%
#     select(starts_with("substance_"), -substance_unique_label) %>%
#     colSums()

# code output below represents the the output from 5-MeO-dmt
```

```
# A tibble: 12 x 3
   substance_key             count percent
   <chr>                     <int>   <dbl>
 1 substance_cannabis           12    20.3
 2 substance_syrian_rue         12    20.3
 3 substance_mdma                9    15.3
 4 substance_salvia              7    11.9
 5 substance_lsd                 5     8.5
 6 substance_nitrous_oxide       4     6.8
 7 substance_alcohol             3     5.1
 8 substance_ketamine            2     3.4
 9 substance_mushrooms           2     3.4
10 substance_ayahuasca           1     1.7
11 substance_methamphetamine     1     1.7
12 substance_unknown             1     1.7
```

In [120]:

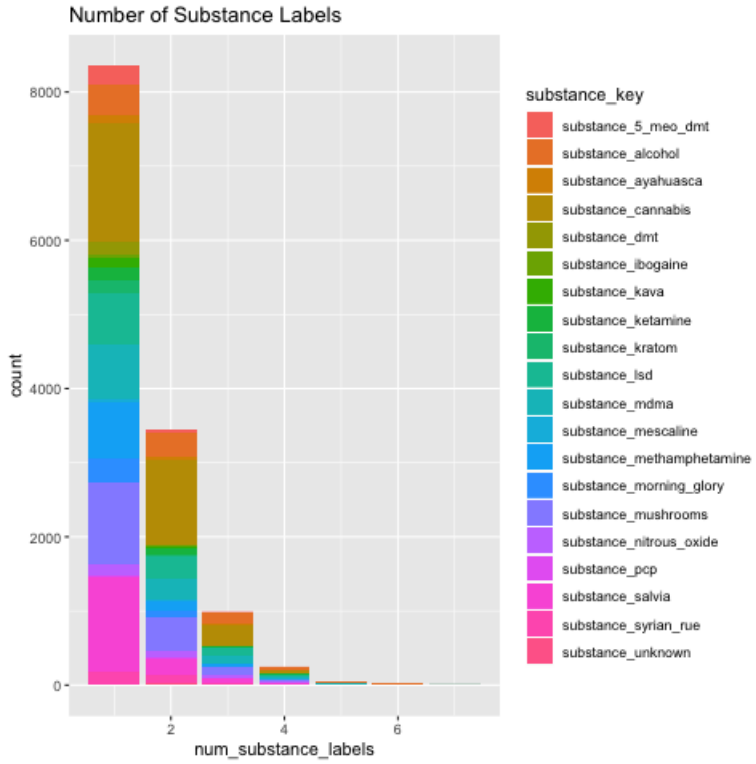
```
%%R
# SUMMARY STATISTICS FOR NUMBER OF SUBSTANCES CONSUMED WITH EACH SUBSTANCE

# convert data to long data format
R_trips_encoded_normalized_eda_long <- R_trips_encoded_normalized_eda %>%
tidyr::gather(substance_key, isPresent, substance_mushrooms:substance_UNK)

## validate the long data format
# R_trips_encoded_normalized_eda_long %>% select(-report) %>% group_by(title) %>% summarize(n()) %
>% head()

# EXPORT:
# Distribution of number of substances for each of the selected substances
R_trips_encoded_normalized_eda_long %>%
    filter(isPresent == 1)  %>%
    group_by(substance_key) %>% summarize(min_num_lab = min(num_substance_labels),
                                          med_num_lab = median(num_substance_labels),
                                          mean_num_lab = mean(num_substance_labels),
                                          max_num_lab = max(num_substance_labels))
```

```
# A tibble: 20 x 5
   substance_key             min_num_lab med_num_lab mean_num_lab max_num_lab
```

	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	substance_5_meo_dmt	1	1	1.20	4
2	substance_alcohol	1	2	1.83	7
3	substance_ayahuasca	1	1	1.44	6
4	substance_cannabis	1	1	1.62	7
5	substance_dmt	1	1	1	1
6	substance_ibogaine	1	1	1.14	3
7	substance_kava	1	1	1.30	5
8	substance_ketamine	1	1	1.60	7
9	substance_kratom	1	1	1.24	5
10	substance_lsd	1	1	1.54	7
11	substance_mdma	1	1	1.55	7
12	substance_mescaline	1	1	1.70	6
13	substance_methamphetamine	1	1	1.27	7
14	substance_morning_glory	1	1	1.29	6
15	substance_mushrooms	1	1	1.46	7
16	substance_nitrous_oxide	1	1	1.79	5
17	substance_pcp	1	2	1.60	4
18	substance_salvia	1	1	1.24	6
19	substance_syrian_rue	1	1	1.54	4
20	substance_unknown	1	2	1.91	4

In [186]:

```
%%R

# DISTRIBUTION OF THE NUMBER OF SUBSTANCE LABELS

# Those without a substance label indicate the substances that were not analyzed
# As we can see, a significant portion of the corpus contain substances that were not analyzed
R_trips_encoded_normalized_eda_long %>%
    filter(isPresent == 1) %>%
    ggplot(aes(x = num_substance_labels, fill = substance_key)) +
        geom_bar() +
        ggtitle("Number of Substance Labels")
```



In [444]:

```
%%R

# PERCENTAGE OF REPORTS BY SUBSTANCE WITH CERTAIN NUMBER OF SUBSTANCES

# https://stackoverflow.com/questions/24576515/relative-frequencies-proportions-with-dplyr
R_trips_encoded_normalized_eda_long %>%
    filter(isPresent == 1)  %>%
```

```
    group_by(substance_key, num_substance_labels) %>%
    summarise(count = n()) %>%
    mutate(percent = round(count / sum(count) * 100, digits=1)) %>%
    filter(num_substance_labels <=3) %>%
    print(n = 100)

# summarize(one_substance = sum(value[num_substance_labels == 1])
```

```
# A tibble: 58 x 4
# Groups:   substance_key [20]
   substance_key             num_substance_labels count percent
   <chr>                                    <dbl> <int>   <dbl>
 1 substance_5_meo_dmt                          1   250    84.2
 2 substance_5_meo_dmt                          2    37    12.5
 3 substance_5_meo_dmt                          3     8     2.7
 4 substance_alcohol                            1   409    44.1
 5 substance_alcohol                            2   323    34.8
 6 substance_alcohol                            3   151    16.3
 7 substance_ayahuasca                          1   114    66.3
 8 substance_ayahuasca                          2    45    26.2
 9 substance_ayahuasca                          3    10     5.8
10 substance_cannabis                           1  1609    51.7
11 substance_cannabis                           2  1162    37.4
12 substance_cannabis                           3   277     8.9
13 substance_dmt                                1   167   100
14 substance_ibogaine                           1    39    90.7
15 substance_ibogaine                           2     2     4.7
16 substance_ibogaine                           3     2     4.7
17 substance_kava                               1   131    78.4
18 substance_kava                               2    24    14.4
19 substance_kava                               3    11     6.6
20 substance_ketamine                           1   170    58.8
21 substance_ketamine                           2    84    29.1
22 substance_ketamine                           3    22     7.6
23 substance_kratom                             1   171    82.6
24 substance_kratom                             2    27    13
25 substance_kratom                             3     6     2.9
26 substance_lsd                                1   696    61.5
27 substance_lsd                                2   304    26.9
28 substance_lsd                                3    96     8.5
29 substance_mdma                               1   747    62.9
30 substance_mdma                               2   288    24.2
31 substance_mdma                               3   111     9.3
32 substance_mescaline                          1    44    60.3
33 substance_mescaline                          2    18    24.7
34 substance_mescaline                          3     6     8.2
35 substance_methamphetamine                    1   746    80.3
36 substance_methamphetamine                    2   135    14.5
37 substance_methamphetamine                    3    32     3.4
38 substance_morning_glory                      1   328    76.8
39 substance_morning_glory                      2    80    18.7
40 substance_morning_glory                      3    14     3.3
41 substance_mushrooms                          1  1094    64.9
42 substance_mushrooms                          2   457    27.1
43 substance_mushrooms                          3   105     6.2
44 substance_nitrous_oxide                      1   155    51.7
45 substance_nitrous_oxide                      2    79    26.3
46 substance_nitrous_oxide                      3    43    14.3
47 substance_pcp                                1    38    46.9
48 substance_pcp                                2    38    46.9
49 substance_pcp                                3     4     4.9
50 substance_salvia                             1  1270    81.6
51 substance_salvia                             2   213    13.7
52 substance_salvia                             3    56     3.6
53 substance_syrian_rue                         1   164    56
54 substance_syrian_rue                         2   103    35.2
55 substance_syrian_rue                         3    22     7.5
56 substance_unknown                            1    13    27.7
57 substance_unknown                            2    27    57.4
58 substance_unknown                            3     5    10.6
```

4.3.4 Text Parsing: Language Syntax and Structure

Text parsing to obtain information about language syntax and structure is often performed as part of the NLP pipeline. We mention

this here for completeness, but omit analysis for the thesis. Interested readers are encouraged to refer to the excellent article A Practioner's Guide to Natural Language Processing). Topics include:

- POS tagging: What are the top nouns, verbs, adjectives etc. associated with each substance?
- Shallow parsing or chunking
- Constituency Parsing
- Dependency Parsing

4.3.5 Text Parsing: Named Entity Recognition

Named Entities refer to real world objects such as an organization, person, quantity, time that are either physical or abstract. An example of a Named Entity is `Apple (ORG)`, the company. The task of Named Entity Recognition (NER) is the process of extracting and labelling these real world objects from free text, and is a non-trivial task — for example, depending on the context, `Apple` can be a proper noun or simply a fruit.

We can use pretrained models from NLP packages such as `spacy` to perform NER and visualize the results. An example is shown below:

In [203]:

```
import spacy
from spacy import displacy
import en_core_web_sm

# load the pretrained small english language model
spacy_nlp = en_core_web_sm.load()
target_report = pd_trips_encoded_normalized.iloc[1777]["report"]

spacy_doc = spacy_nlp(target_report)
displacy.render(spacy_doc, style="ent", jupyter = True)
```

going middle school part high school , always un ORG -happy weight. ran cross-country , always hungry. saw advertisement magazine adderall , saw main side effect loss appetite. asked parents getting tested add said no. parents divorced , asked mom said could get tested. made appointment doctor , doctor told would go specialist get testing. went , acted ' add ' possible. sure enough , diagnosed add. , excited finally get it. normal doctor wanted put strattera instead adderall , fight 2 months DATE get it. got adderall , gave 25 mg QUANTITY pills. absolutly love adderall. first ORDINAL time took 8 p.m. TIME , wanted help homework (stupid move) . snorted 1 CARDINAL pill , took 2 CARDINAL normally. took much heard someone first ORDINAL time ever type speed best effect ' cause body hasnt ever anything like , wanted make really good. holy shit ever. finished homework like 10 min. , cleaned entire room. cleaned bathroom , basement. came upstairs 3 CARDINAL started getting vacuum wanted vacuum living room , didnt cause didnt want wake mom. didnt sleep entire night TIME , next day DATE around 8 , took another wasnt tired throughout day. happy awake ! would take morning TIME , smile face no reason. normally would walk first class day , would tired sit down. adderall GPE , would energetic happy , talk people outside class. made confident positive. never ate either. could go 6 7 days DATE without eating single meal. would finally break eat something. put adderall , around 175 CARDINAL lbs. 5 ' 10 ' ' , not obese NORP , insulated will. weeks taking adderall , noticed 1 pill day DATE wasnt good used be. started taking 2 3 day DATE . sometimes would snort morning TIME felt like it. started running really early , one make appointment doctor , mom never came me. doctor stupid never noticed appointments 2 weeks earlier DATE been. started loosing lot weight. 3 months DATE taking adderall , 140 CARDINAL lbs. happy ever. confident , everyone complimented good looked. notice getting little strung though , rarely sleeping , adderall helping less less , even though taking 4 CARDINAL pills day (100 mg QUANTITY /day) . always got really bad dry-mouth , would drink 6 CARDINAL water bottles school (water bottle 32 CARDINAL oz. thats like 192 oz day) . also really dependent , would depressed didnt take it. doctor soon noticed weight loss flipped said couldnt believe let happen get far shit like that. told didnt gain 10 CARDINAL lbs 2 weeks DATE , stopping

treatment completely. no way going gain weight , didnt got pissed. put concerta , refuse take. hate effects gives me. feel sad introverted. doesnt help concentrate , like take pill feel like shit really sad , no thanks. kid know told sister used take ritalin stopped , still ton house ORG . interested (course) sold bottle 20 CARDINAL , 20mg TIME . adderall 60 CARDINAL , 10 mg QUANTITY ritalin 5$ MONEY (haha PERSON good deal !) . took adderall , started ritalin. always thought ritalin rush adderall , better. wasnt all. didnt feel anything really , except snorted it. felt slightly ' ' 20 ish min. , like really weak line coke. didnt help w/ appetite either. friend asked wanted try coke ORG , it. bought gram PERSON shared 3 CARDINAL us. guess really bad coke cause didnt much me. pupils got really dilated , started sweating , little energetic , it. people always said coke ORG amazing wonderful , wasnt me. bummed PERSON , got another gram PERSON couple days later DATE see would better time. wasnt friends found got mad , whole situation fucked up. kinda PERSON gave coke ORG wasted lot money source really shitty coke PRODUCT . since gained back weight im really sad it. without adderall , feel slow , stupid , depressed. taking friends step-mom drugs said ' never wanted knew would like them- people dont shoot heroin hurts , feels fuckin NORP good. know would like much , never even try it. ' thought cool know always want kind thing speed up. boring normal .

4.3.6 Text Parsing: Sentiment Analysis

Another task that is common in the NLP pipeline is to perform sentiment analysis with pre-trained models. The most commonly used models (source) are

- AFINN lexicon
- Bing Liu's lexicon
- MPQA subjectivity lexicon
- SentiWordNet
- VADER lexicon
- TextBlob lexicon

We include this for completeness, but skip the in-depth analysis for this thesis.

4.3.7 Treatment of the Longtail

As mentioned, there were a substantial number of substance in the corpus, and it was unfeasible to analyze all the substances. In the end we decided to focus on 20 labels, which was chosen both for their pharmalogical properties (psychedelics or most closely resembling psychedelics), and also their frequencies. All the substances chosen in the following list were among the most frequently occuring substances in the corpus

- substance.mushrooms
- substance.lsd
- substance.mescaline
- substance.cannabis
- substance.mdma
- substance.ayahuasca
- substance.nitrous_oxide
- substance.salvia
- substance.methamphetamine
- substance.dmt
- substance.5_meo_dmt
- substance.alcohol
- substance.ketamine
- substance.ibogaine
- substance.pcp
- substance.kava
- substance.kratom
- substance.morning_glory
- substance.syrian_rue
- substance.unknown

In [2]:

In [2]:

```
# create a constant list to reprsent chosen substances
SUBSTANCES_OF_INTEREST_LIST = ['substance_5_meo_dmt',
 'substance_alcohol',
 'substance_ayahuasca',
 'substance_cannabis',
 'substance_dmt',
 'substance_ibogaine',
 'substance_kava',
 'substance_ketamine',
 'substance_kratom',
 'substance_lsd',
 'substance_mdma',
 'substance_mescaline',
 'substance_methamphetamine',
 'substance_morning_glory',
 'substance_mushrooms',
 'substance_nitrous_oxide',
 'substance_pcp',
 'substance_salvia',
 'substance_syrian_rue']

# the most classical psychedelics
SUBSTANCES_PSYCHEDELICS_LIST = ["substance_mushrooms",
                                "substance_lsd",
                                "substance_mescaline",
                                "substance_5_meo_dmt",
                                "substance_dmt"]
```

4.4 The Data: Clean and Filtered

The following data frame represents the data that has been cleaned, filtered, and encoded, and will be used in all subsequent analysis.

In [0]:

```
# Saving the cleaned, filtered, and encoded data to file
# pd_trips_encoded_normalized.to_csv("pd_trips_encoded_normalized.csv", index = False)
```

In [5]:

```
# README: Use the following import to load the cleaned data back into memory
# EXPORT:
import pandas as pd

pd_trips_encoded_normalized = pd.read_csv("pd_trips_encoded_normalized.csv")
```

In [6]:

```
# first 6 rows of the cleaned data
pd_trips_encoded_normalized.head(n=6)
```

Out[6]:

	Unnamed: 0	report	title	substance	substance_mushrooms	substance_lsd	substance_mescaline	substance_cannabis
0	1	success forms legal highs , decided experiment...	Sideways World	salvia divinorum (5x extract)	0	0	0	0
1	2	couple buddies decided one night past february...	Physical Wellbeing = Crucial	mushrooms	1	0	0	0
2	3	girlfriend saving methylone special occasion ,...	The Artful Dodger	methylone	0	0	0	0

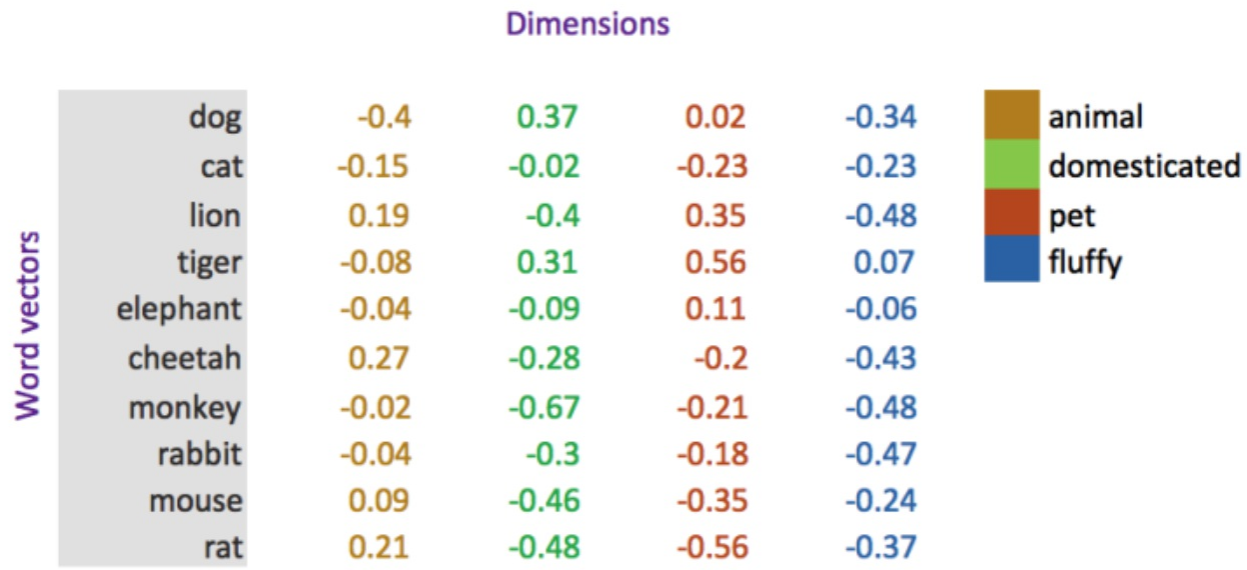
	Unnamed: 0	report	title	substance	substance_mushrooms	substance_lsd	substance_mescaline	substance_cannabis
3	4	want warn anybody taking lithium (probably ps...	Seizure Inducing Combo	lsd & lithium	0	1	0	0
4	5	several attempts breakthrough 5-meo-dmt. belie...	Enlightenment Through a Chemical Catalyst	5-meo-dmt	0	0	0	0
5	6	interesting experience salvia nights ago want ...	Park Overhangs and Dog Hair	salvia divinorum (6x extract)	0	0	0	0

6 rows × 26 columns

5. Text Presentation and Feature Engineering

5.1 Word as Vectors

For those unfamiliar with natural language processing, we introduce a simple but very central and powerful idea: ***each `word` can be represented as a vector of real numbers***. This may seem counterintuitive, but the ability to represent words as vectors is absolutely key to our ability to perform analysis on unstructured text — almost all machine learning algorithms take vectors or matrixes as inputs and vector representations open us to a world of vector and matrix operations such as dot products and cosine similarity, which allow us to ask (and answer) questions such as `how **similar** are two words?` intelligently and precisely.



`Infographic Source`

In the above infographic, each word is represented by 4 numbers, or 4 dimensions, representing a word's `animal`-ness, `domesticated`-ness, `pet`-ness, `fluffy`-ness. With this, we can ask whether a monkey is more similar to an elephant or more similar to a cheetah using cosine similiarity.

In [0]:

```
# Illustration of cosine similarity
# It looks like a monkey is more similar to a cheetah than it is to an elephant!
monkey = np.array([-0.02, -0.67, -0.21, -0.48])
elephant = np.array([-0.04, -0.09, 0.11, -0.06])
cheetah = np.array([0.27, -0.28, -0.2, -0.43])

# cosine similarity relies on the intuition that the smaller the angle between two vectors, the mo
re similar they are
def cos_sim(v1, v2):
    sim = np.dot(v1, v2) / (np.sqrt(sum(v1**2)) * np.sqrt(sum(v2**2)))
    return sim

cos_sim(monkey, elephant) # 0.4926634171010775
cos_sim(monkey, cheetah) # 0.8251926299381945
```

Out[0]:

```
0.8251926299381945
```

Words are represented as vectors mainly in two ways (see here also):

1. count of word / context co-occurences
2. vectors that naturally arise from trying to predict the context of a word (GloVE, Skipgram models)

The first class of methods relies on the distributional hypothesis in linguistic (Zellig Sabbettai Harris, 1954), which formalized the intuition that *words that co-occur together in the same contexts tend to be more similar*. There are two main ways to construct vectors this way:

- **`term-document matrix`**: Very briefly, suppose we had a vocabulary (unique tokens) of size V, and a collection of documents (such as wikipedia pages) of size D. We can make a V x D matrix `TDM`, with words as the rows and documents as the columns. The position M(i, j) would then be how many times word i occurs in document j. Just like that, we have created a *term-document matrix*, one of the most important building blocks in natural language processing (the transpose, document-term matrix, is equally often used). Recall that each row represents a word, and each row is a vector of dimension D. In this precise way, we have just represented each of the V words in our vocabulary as a vector of dimension D.
- **`term-term matrix`**: now, instead of using a V X D matrix, we can make a V x V matrix `TTM`, and figure out a sensible way to populate the elements. One intuitive way is to take a "sliding window" with some width over the text, and count how many times some word j occurs in the context window of word i. That count would then go into TTM(i, j). In this way, each word is represented as V dimensional vector.

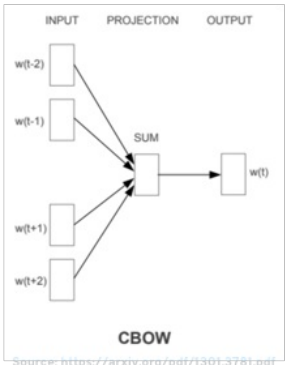
Note that because V and D are often on the order of millions, these vector representations are absolutely huge, and many fields would be 0. For this reason, these are also called ***sparse* vector embeddings**.

The second class of methods rely on the intuition that if we use words as inputs to peform some machine learning task (such as predicting the most likely surrounding context words given a current word using a neural network, say), the weights and biases learned by the neural network will act as a "good" approximations for the vector representation of the word.

The most commonly used models are `Word2Vec` (CBOW and Negative Skip Gram Sampling) by Google, `GloVe Model` from Stanford University, and `FastText Model` by Facebook. Although the three models rely on different machine learning tasks, they all rely on randomly intialized vectors converging to vector approximations of words as the machine learning task trains (Chris Callison-Burch). The exact details are beyond the scope of this thesis, but there are 3 very important takeaways: (1) The dimensions of the word vectors are arbitrary and user specified, (2) The numbers in the learned vectors don't have *intrinsic* meaning like how the ones in the sparse representations above do. The learned vector for a word is an abstract embedding in a higher dimensional space, derived from a particular training corpus and particular algorithm. There is no *single* vector that ***is*** a particular word. Finally, (3) Since the dimensions are user specified they are typically kept to around 100 to 300, and as such these word vectors are often called ***dense vector embeddings***

For those interested, here are three infographics we made that provide some context for the different algorithms, examples of embeddings learned, and links to the original papers that described the different papers. Notice here that words in the `Word2Vec` are 100-dimensional vector embeddings, and words in the `GloVe` example are 300-dimensional vector emeddings. Indeed, multiple pre trained models with different initial configurations (such as dimension of embedding and size of training corpus) are readily available for use. For a deeper yet approachable introduction to word embeddings arising from deep learning models, Diparjan Sarka's article on Towards Data Science is an excellent place to start

Word2Vec Models



Source: https://arxiv.org/pdf/1301.3781.pdf

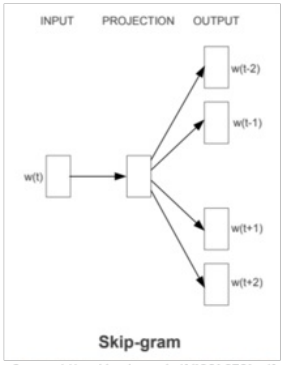
Continuous Bag of Words (CBOW)

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
shall	-1.183386	-2.866214	1.046431	0.943265	-1.021784	-0.047069	2.108584	-0.458692	-1.698881	0.905800	...	0.655786	0.703828	0.821803	-0.093732
unto	-1.725262	-1.765972	1.411971	0.917713	0.793832	0.310631	1.541964	-0.082523	-1.346811	0.095824	...	1.682762	-0.872293	1.908597	0.977152
lord	1.694633	-0.650949	-0.095796	0.950002	0.813837	1.538206	1.125482	-1.655581	-1.352673	0.409504	...	1.553925	-0.819261	1.086127	-1.545129
thou	-1.590623	-0.801968	1.659041	1.314925	-0.455822	1.733872	-0.233771	-0.638922	0.104744	0.490223	...	0.652781	-0.362778	-0.190355	0.040719
thy	0.386488	-0.834605	0.585985	0.801969	-0.165132	0.999917	1.224088	-0.317555	-0.671106	-1.073181	...	1.267184	-0.564660	0.089618	-0.979835

5 rows × 100 columns

Word Embeddings for our vocabulary based on the CBOW model

Source: https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa



Source: https://arxiv.org/pdf/1301.3781.pdf

Skip-gram

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
shall	-0.002272	0.015870	0.018349	0.022802	0.028364	-0.040064	-0.013263	0.136607	0.019667	0.033407	...	0.037663	-0.087140	0.073169	-0.028257
unto	0.034425	-0.102070	0.018051	0.017960	0.172954	-0.115672	-0.012632	0.096919	-0.049203	-0.040344	...	0.106373	-0.075703	0.013888	-0.134224
lord	0.051990	-0.113865	0.007226	0.031754	0.052963	-0.094523	-0.067664	0.001706	-0.112827	-0.078586	...	-0.041636	0.053685	0.041299	-0.026255
thou	-0.152183	-0.073681	-0.091472	0.022033	0.008415	-0.048438	-0.041181	0.082019	0.004648	0.044870	...	0.101531	-0.018404	-0.070462	-0.041363
thy	-0.257579	-0.023008	0.053303	0.013690	-0.083293	0.034279	0.078811	0.079851	-0.015215	-0.111211	...	-0.064527	0.112085	0.061625	0.026398

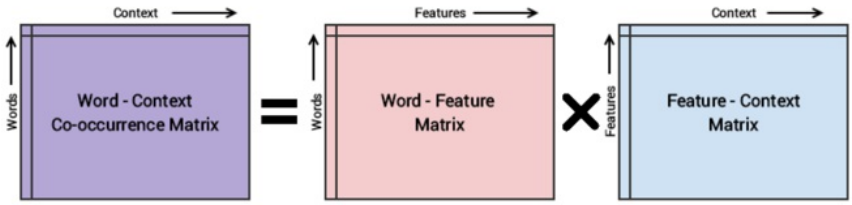
5 rows × 100 columns

Word Embeddings for our vocabulary based on the Skip-gram model

Source: https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa

GLoVe Model

GLoVe = Global Vectors for Text Representation



Conceptual model for the GloVe model's implementation

sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...	-0.667050	0.279110	0.500970	-0.27
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...	0.304240	0.413440	-0.540730	-0.03
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...	0.647710	0.373820	0.019931	-0.03
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...	0.142080	0.481910	0.045167	0.05
today	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...	-0.326580	-0.413380	0.367910	-0.26
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...	-0.501280	0.169010	0.548250	-0.31
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...	-0.405170	0.243570	0.437300	-0.46
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	...	-0.470090	0.063743	-0.545210	-0.19
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...	0.003257	-0.036864	-0.043878	0.00
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...	0.302240	0.195470	-0.653980	-0.29
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	...	0.402310	-0.038554	-0.288670	-0.24
love	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...	-0.124380	0.178440	-0.099469	0.00
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...	-0.329460	0.421860	-0.039543	0.15

20 rows × 300 columns

GloVe embeddings for words in our toy corpus

GloVe: Global Vectors for Word Representation

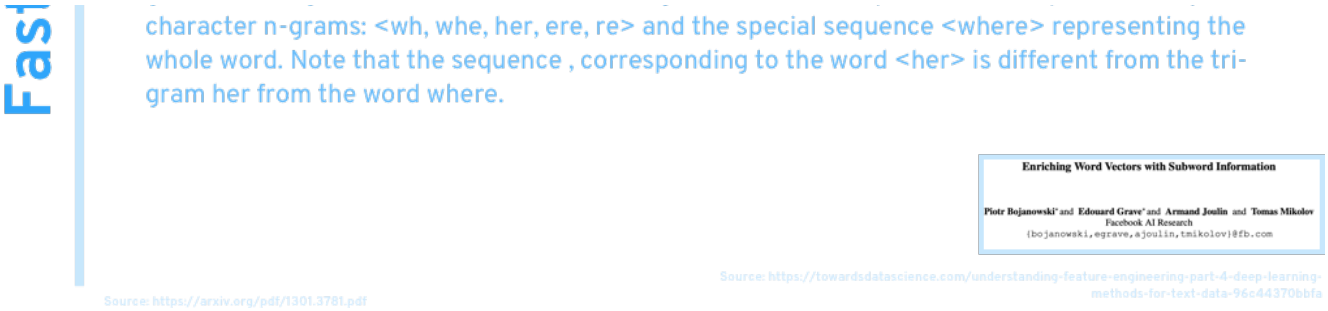
Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

Source: https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa

t Text Model

In general, predictive models like the Word2Vec model typically considers each word as a distinct entity (e.g. where) and generates a dense embedding for the word. However this poses to be a serious limitation with languages having massive vocabularies and many rare words which may not occur a lot in different corpora. The Word2Vec model typically ignores the morphological structure of each word and considers a word as a single entity. **The FastText model considers each word as a Bag of Character n-grams. This is also called as a subword model in the paper.**

We add special boundary symbols < and > at the beginning and end of words. This enables us to distinguish prefixes and suffixes from other character sequences. We also include the word w itself in the set of its n-grams, to learn a representation for each word (in addition to its character n-grams). Taking the word where and n=3 (tri-grams) as an example, it will be represented by the



5.2 Documents as Vectors

Once the idea that words can be represented as vectors sinks in, the idea that entire documents can be represented as vectors feel very intuitive. Recall in the previous section we saw a **`term-document matrix`**, which was a V x D matrix with terms as rows and documents as columns. Taking the rows, we represented each term as a D-dimensional vector. On the flip side, if we simply took the columns, we could represent documents as V dimensional vectors. This results in a **sparse embedding** of a document.

We can also utilize the fact that documents are a collection of words, each of which can be represented by a vector. As such, we can simply take the average of all the word vectors in a document, and call that our document vector. This results in a **dense embedding** that represents a document. In practice, this seemingly naive method of averaging works surprisingly well (Chris Callison-Burch), and the intuition of averaging word vectors is agnostic to which word embedding was chosen, be it `GloVe`, `Word2Vec`, `FastText` or a DIY embeddng. One commonly used document embedding is Google's `Doc2Vec`, which, as you might have guessed, builds on top of `Word2Vec` word embeddings.

5.3 Vectorizing our Corpus with Bag of Word (BOW) Features

Now that we have introduced the important intuition that words and documents can be represented as vectors, we can bring this theory into practice and vectorize our corpus. Let us begin with the basics and slowly get more complex.

5.3.1 Term-Document Counts: The Very Basics

This corresponds to constructing the aforementioned `term-document matrix`, where we constructed a V x D matrix from the corpus, where each document is represented with a V dimensional vector. `sklean` has a built in vectorzier called `CountVectorizer` which we can use directly on our normalized corpus. As an implementation detail, this procedure actually produces a **`document-term matrix`**, which is the aforementioned `term-document matrix` flipped along the diagnal. Practically, this creates a D x V matrix with documents as rows and terms as columns, with all else the same. Moving forward, we will adopt the convention of keeping documents as rows and terms as columns.

In [447]:

```
# OPTIONAL: In case Kernel dies, use this to load back dataset
import pandas as pd
pd_trips_encoded_normalized = pd.read_csv('{}/pd_trips_encoded_normalized.csv'.format(root_path))
# with open('pd_trips_encoded_normalized.csv', 'rb') as in_file:
#     pd_trips_encoded_normalized = pickle.load(in_file)
# pd_trips_encoded_normalized.head()
```

Out[447]:

	Unnamed: 0	report	title	substance	substance_mushrooms	substance_lsd	substance_mescaline	substance_cannabis
0	1	success forms legal highs , decided experiment...	Sideways World	salvia divinorum (5x extract)	0	0	0	0
1	2	couple buddies decided one night past february...	Physical Wellbeing = Crucial	mushrooms	1	0	0	0
2	3	girlfriend saving methylone special occasion ,...	The Artful Dodger	methylone	0	0	0	0
		want warn						

	Unnamed: 0	report	title	substance	substance_mushrooms	substance_lsd	substance_mescaline	substance_cannabis
3	4	anybody taking report (probably ps...	Seizure Inducing Combo	lsd & lithium	0	1	0	0
4	5	several attempts breakthrough 5-meo-dmt. belie...	Enlightenment Through a Chemical Catalyst	5-meo-dmt	0	0	0	0

5 rows × 26 columns

In [449]:

```
# SPIKE: for some very strange reason, reading the csv file back resulted in np.nan trip reports
pd_trips_encoded_normalized["report"].isnull().sum()
# dropna: https://stackoverflow.com/questions/13413590/how-to-drop-rows-of-pandas-dataframe-whose-value-in-a-certain-column-is-nan
pd_trips_encoded_normalized.dropna(subset=['report'], inplace=True)
# check there are no NA's
# pd_trips_encoded_normalized["report"].isnull().sum()
```

In [450]:

```
# Code adapted from:
# https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41
from sklearn.feature_extraction.text import CountVectorizer

# CountVectorizer expects a list of documents; i.e.: [string]
norm_corpus = pd_trips_encoded_normalized["report"].to_list()
len(norm_corpus) # 19924 reports

# instead of using raw counts, we can divide matrix(i, j) by sum over column j to get a proporition
# min_df = minimum document frequency
# max_df = maximum document frequency
tf_vectorizer = CountVectorizer(min_df=0., max_df=1.)
dtm_tf = tf_vectorizer.fit_transform(norm_corpus)
dtm_tf_nparr = dtm_tf.toarray()

# get all unique words in the corpus
vocab = tf_vectorizer.get_feature_names()

# show document feature vectors
pd.DataFrame(np.round(dtm_tf_nparr, 2), columns=vocab).head()

##############################################
## Ngram model extensions
# you can set the n-gram range to 1,2 to get unigrams as well as bigrams
# bv = CountVectorizer(ngram_range=(2,2))
# bv_matrix = bv.fit_transform(norm_corpus)

# bv_matrix = bv_matrix.toarray()
# vocab = bv.get_feature_names()
# pd.DataFrame(bv_matrix, columns=vocab)
```

Out[450]:

	00	000	0000	00000	00000000	000000000000000000000000000003	00005	0001	0001g	0002	...	zzzs	zzzzang	zzzzz	zzzzzz	zzz
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 107108 columns

In [451]:

```
# Some basic properties we can learn about our data
len(vocab) # 107108 vocabulary words, or "tokens", in our training corpus
vocab[20000:20010] # example of tokens
```

Out[451]:

```
['clasify',
 'clasp',
 'clasped',
 'clasping',
 'clasps',
 'class',
 'classa',
 'classed',
 'classes',
 'classic']
```

Each row represents a document, and each column is a vocabulary word. We notice that the term-document matrix generated is extremely sparse (mostly zeros), and there seems to be a large number of non-sense tokens such as `00`, `000`, `0000` etc. More processing may be required to remove these tokens for better performance.

5.3.2 TF-IDF Weighting Scheme

Using just the raw word frequencies, we might overemphasize words that are shared across all trip reports, giving us little if any insight into the most significant words that is associated with each type of substance. If some word is faily rare (like "avocado") in general in the English language, but appeared quite few times in a single document, it makes intuitive sense to upweight the importance of that rare word with respect to that particular document. In other words, we need a formlua to capture the intuition "if a word appears very frequently in a particular document and doesn't appear too frequently in other documents, the word is probably important for that particular document". Capturing this intuition and to improve upon the raw frequency baseline, we can use the `tf-idf` weighting for each (word, document) pair as a weighting term.

`tf-idf` stands for `term frequency-inverse document frequency`, and is specified for each (word, document) pair. It is by far one of the most widely used weighting scheme in considering how important a word is to a document, and is widely used in text mining and information retrieval. For an approachable academic introduction to `tf-idf`, see Chris Callison-Burch's lecture slides.

- `tf` stands for `Term Frequency` (Luhn 1957), and is simply the number of times a word appears in a document. Precisely, the term trequency of word i in document j is given by: tf_{ij} = # of times word i appears in document j
- `idf` stands for `Inverse Document Frequency` (Spark Jones 1972), and intuitively measures how many documents contain a particular word, with words appearing frequently in many documents having less importance to a particular document, thus having a lower `idf` score. Precisely, the inverse document frequency of word i is given by

$$idf_i = log\left(\frac{N}{df_i}\right)$$

Where df_i is the `document frequency of word i`, the number of documents containing word i

With these definitions, we stay the tf-idf weighting of word i in document j is given by

$$tf.idf(i,j) = tf_{ij} \cdot idf_i$$

In [453]:

```
# TFIDF Code
# adapted from: https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
dtm_tfidf = tfidf_vectorizer.fit_transform(norm_corpus)
dtm_tfidf_nparr = dtm_tfidf.toarray()

vocab = tfidf_vectorizer.get_feature_names()
pd.DataFrame(np.round(dtm_tfidf_nparr, 2), columns=vocab).tail()
```

Out[453]:

	00	000	0000	00000	00000000	0000000000000000000000000003	00005	0001	0001g	0002	...	zzzs	zzzzang	zzzzz	zzzzzz
19910	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
19911	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
19912	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
19913	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
19914	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

5 rows × 107108 columns

In [42]:

```
# OPTIONAL
## Use this cell as a template to save and load large data files;
## replace filenames appropriately;

# import pickle

# ### Stores the tokenized reports:
# with open('dtm_tfidf', 'wb') as out_file:
#     pickle.dump(dtm_tfidf, out_file)
```

5.3.3 Document Similarity matrix: Bag of Words

We can use these vectorized representations of documents to ask: how similar are two documents? Or rather, how similar are any two trip reports to each other? With this question in mind, we can construct a similiarity matrix based on cosine similarity. Now, the following matrix is 19924 x 19924, and is very hard to visualize and interpret. Nevertheless, we include it here for completeness as one way to visualize the relationship between documents.

Similarity matrices such as these are building block inputs to clustering algorithms such as KMeans and certain visualization algorithms (See sklearn), and is an important component in the data scientist's toolkit.

In [398]:

```
# Adapted from https://towardsdatascience.com/understanding-feature-engineering-part-3-
traditional-methods-for-text-data-f6f7d70acd41
from sklearn.metrics.pairwise import cosine_similarity

similarity_matrix_tfidf = cosine_similarity(dtm_tfidf)
similarity_tfidf_df = pd.DataFrame(similarity_matrix_tfidf)
similarity_tfidf_df.head(10)
```

Out[398]:

	0	1	2	3	4	5	6	7	8	9	...	19905	19906	
0	1.000000	0.129403	0.090606	0.076207	0.131606	0.182589	0.127793	0.053275	0.083754	0.062670	...	0.066514	0.104057	0.1
1	0.129403	1.000000	0.074791	0.069993	0.112724	0.083869	0.096765	0.066576	0.069284	0.052974	...	0.045412	0.074807	0.0
2	0.090606	0.074791	1.000000	0.054994	0.082718	0.056025	0.065638	0.033547	0.091073	0.075035	...	0.035412	0.086487	0.0
3	0.076207	0.069993	0.054994	1.000000	0.056029	0.057267	0.083636	0.043510	0.053084	0.025352	...	0.022614	0.068686	0.0
4	0.131606	0.112724	0.082718	0.056029	1.000000	0.101215	0.109337	0.032008	0.077434	0.075994	...	0.039520	0.072239	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
19910	0.099802	0.060976	0.061073	0.063296	0.044890	0.066234	0.056905	0.045736	0.072816	0.065657	...	0.070000	0.069395	0.0
19911	0.108131	0.071558	0.039170	0.061351	0.053280	0.056859	0.082844	0.025499	0.059631	0.047284	...	0.025946	0.045970	0.0
19912	0.114546	0.081604	0.097778	0.061448	0.096490	0.111927	0.101830	0.042317	0.086267	0.081419	...	0.067819	0.099276	0.0
19913	0.249854	0.134838	0.117509	0.083141	0.153801	0.181383	0.145912	0.060385	0.089602	0.094643	...	0.080443	0.113990	0.1
19914	0.139503	0.124417	0.095866	0.091039	0.105303	0.093093	0.148827	0.039383	0.085033	0.049296	...	0.054736	0.073996	0.1

19915 rows × 19915 columns

In [0]:

```
# OPTIONAL
# with open('similarity-matrix_dtm_tfidf', 'wb') as out_file:
#     pickle.dump(similarity_tfidf_df, out_file)
```

6. Topic Modeling with Latent Dirichlet Allocation (LDA)

Intuitively, one can imagine that a document such as a blog post, wikipedia article, or in our case, a psychedelic trip report, might consist of numerous `topics` — a news article published in the New York Times might talk about extreme weather, deforestration, and climate change, when covering say, a hurricane; a psychedelic trip report might talk about time, experiences with nature, or feelings of bliss. How do we figure out the topics in an unlabelled collection of documents? We turn to a class of topic modelling algorithms, of which the most widely used is Latent Dirichlet Allocation (LDA). (Side note: the acronym LDA is also often used to mean "Linear Discriminant Analysis", used for classification or dimensionality reduction)

6.1 Latent Dirichlet Allocation: An Introduction

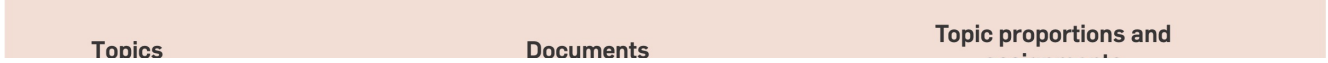
Latent Dirichlet Allocation (LDA) has fancy sounding name, and the inner workings are indeed fairly complex, but what it does is quite intuitive: LDA attempts to (1) automatically discover topics from a collection of documents, and (2) model each document in that collection as exhibiting multiple topics in diffrent proportions (Blei 2012). So what *is* a `topic`? Each `topic` is precisely a list of words. For example, the following lists of words represent 4 topics extracted automatically from the Journal **Science**, where each topic is a list of words (Blei 2012, Fig 2):

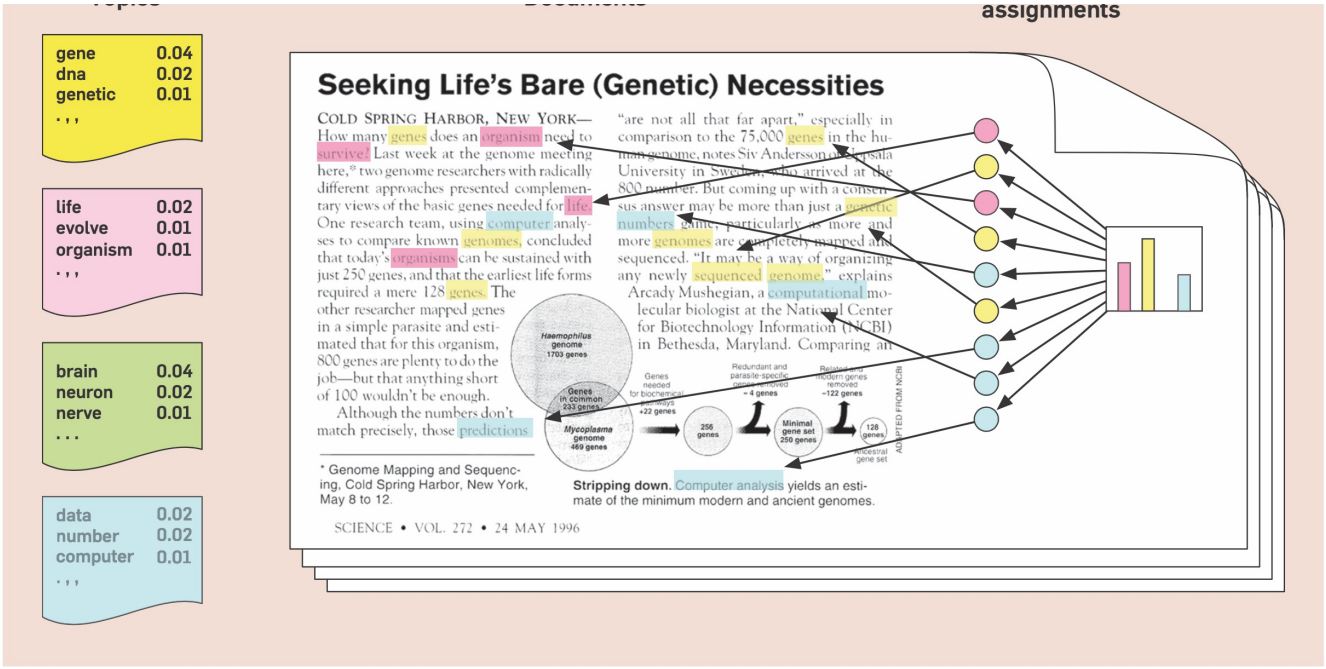
- `topic1`: human genome dna genetic genes sequence gene molecular sequencing map information genetics mapping project sequences
- `topic2`: evolution evolutionary species organisms life origin biology groups phylogenetic living diversity group new two common
- `topic3`: disease host bacteria diseases resistance bacterial new strains control infectious malaria parasite parasites united tuberculosis
- `topic4`: computer models information data computers system network systems model parallel methods networks software new simulations

It should be noted that since LDA is an unsupervised technique, we don't require our input text documents have topic labels. Instead, LDA processes our collection of text documents, and extracts topics automatically. Notice also that the topics discovered don't have "names" by themselves; by looking at the list of words associated with each topic above, it is reasonable to say `topic1` is approximately "genetics", `topic2` is approximately "evoluation", `topic 3` is approximately "disease", and `topic4` is approximately "computers"; in practice the labelling of topics can either by done by humans, or by automatically taking a word that is most "relavent" to that topic as the label, by some sensible definition of "relavence". Indeed, the above word list are sorted from most relavent to least relavent for each topic.

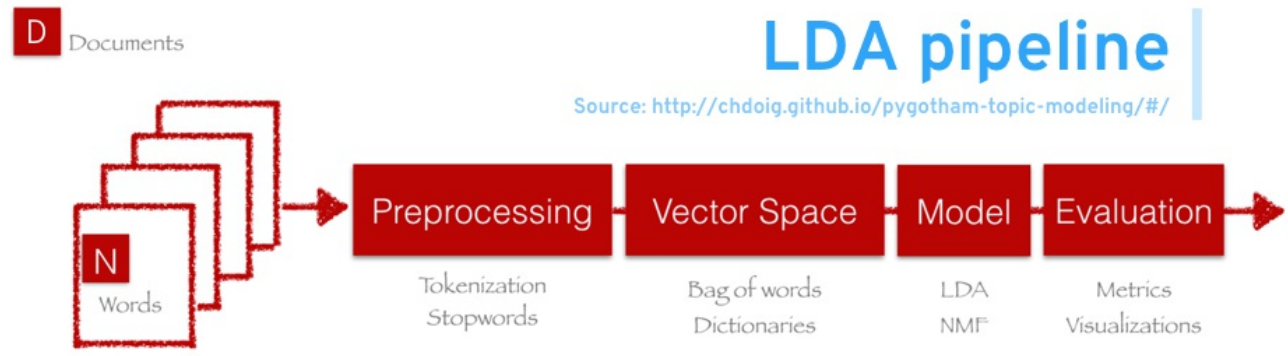
This beautiful graphic presented by Blei 2012 helps us further the intuition of how each document can exhibit multiple topics:

Figure 1. The intuitions behind latent Dirichlet allocation. We assume that some number of "topics," which are distributions over words, exist for the whole collection (far left). Each document is assumed to be generated as follows. First choose a distribution over the topics (the histogram at right); then, for each word, choose a topic assignment (the colored coins) and choose the word from the corresponding topic. The topics and topic assignments in this figure are illustrative—they are not fit from real data. See Figure 2 for topics fit from data.



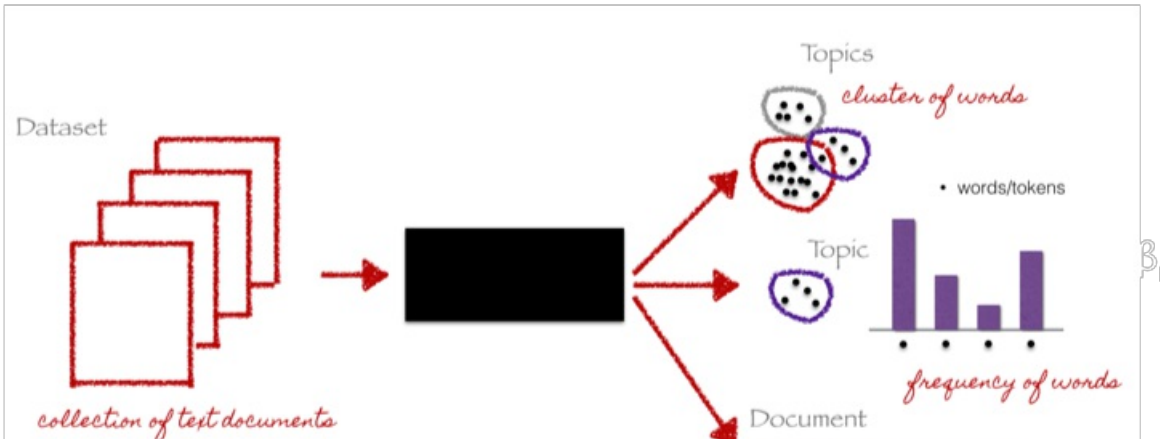


Now, before we dive deeper into the inner workings of LDA, let's take a look at the LDA topic modelling pipeline, lest we lose track of the big picture:



As we can see, we have the collection of D documents (i.e.: trip reports from Erowid), each with some N_j words, which was obtained via processes described in *Section 3: Data Aquisision*. In *Section 4: Data Cleaning and Exploratory Data Analysis (EDA)*, we processed our documents using an array of pre-processing techniques such as tokenization and stopwords removal. In *Section 5: Text Representation and Feature Engineering*, we created vector representations of documents through a variety of methods, including the naive (but fairly effective) term-document counts, along with tf-idf weighting, both of which fall under "Bag of Words (BOW)" methods. And now, in this section, we use LDA to create a topic model to capture the topics representd in our collection of trip reports. It should be noted that LDA is not the only topic model available to modern data scientists. Other methods include Non-negative Matrix Fatorization (NMF) and Probabilistic Latent Semantic Analysis (pLSI), which we note here but do not dive deeply into to maintain thesis scope. For those familiar with singular value decomposition (SVD): both NMF, and pLSI relate deeply to SVD of the term-document matrix. Furthermore, "LDA can also be seen as a type of principal component analysis (PCA) for discrete data" (Blei 2012).

Now, we are ready dive deeper into what LDA does. Let us begin with an infographic for intuition:



6.2 The Input to LDA

LDA takes as input a collection of text documents, which has been preprocessed and vectorized. Note that the format of these text documents isn't set in stone, but most commonly for LDA implementation we have a D x m matrix, where each row represents a document d, represented by a m-dimensional document vector embedding. We discussed many document vector embeddings in Section 5. For ease, we can use the tf-idf weighted term-document counts.

For concreteness, below is the vectorized representation of the first 6 documents in our corpus:

In [0]:

```
# First 5 rows of our tf-idf weighted document-term matrix
# each row represents a document
# each column is a term, or "features"
pd.DataFrame(np.round(dtm_tfidf_nparr, 2), columns=vocab).head()

# wierdly enough, this following does not work; has to do with sparse vs. dense representations
# pd.DataFrame(np.round(dtm_tfidf, 2), columns=vocab).head()
```

Out[0]:

	00	000	0000	00000	00000000	00000000000000000000000000003	00005	0001	0001g	0002	...	zzzs	zzzzang	zzzzz	zzzzzz	zz
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

5 rows × 107108 columns

In [0]:

```
# Using the document-term matrix with tfidf weighting, our input is D x m, where D = 19915, the nu
mber of documents
# and m = 107108, or V, the size of our vocabulary
dtm_tfidf_nparr.shape
```

Out[0]:

```
(19915, 107108)
```

6.3 The Output of LDA

Referring to the picture above, LDA gives us three things:

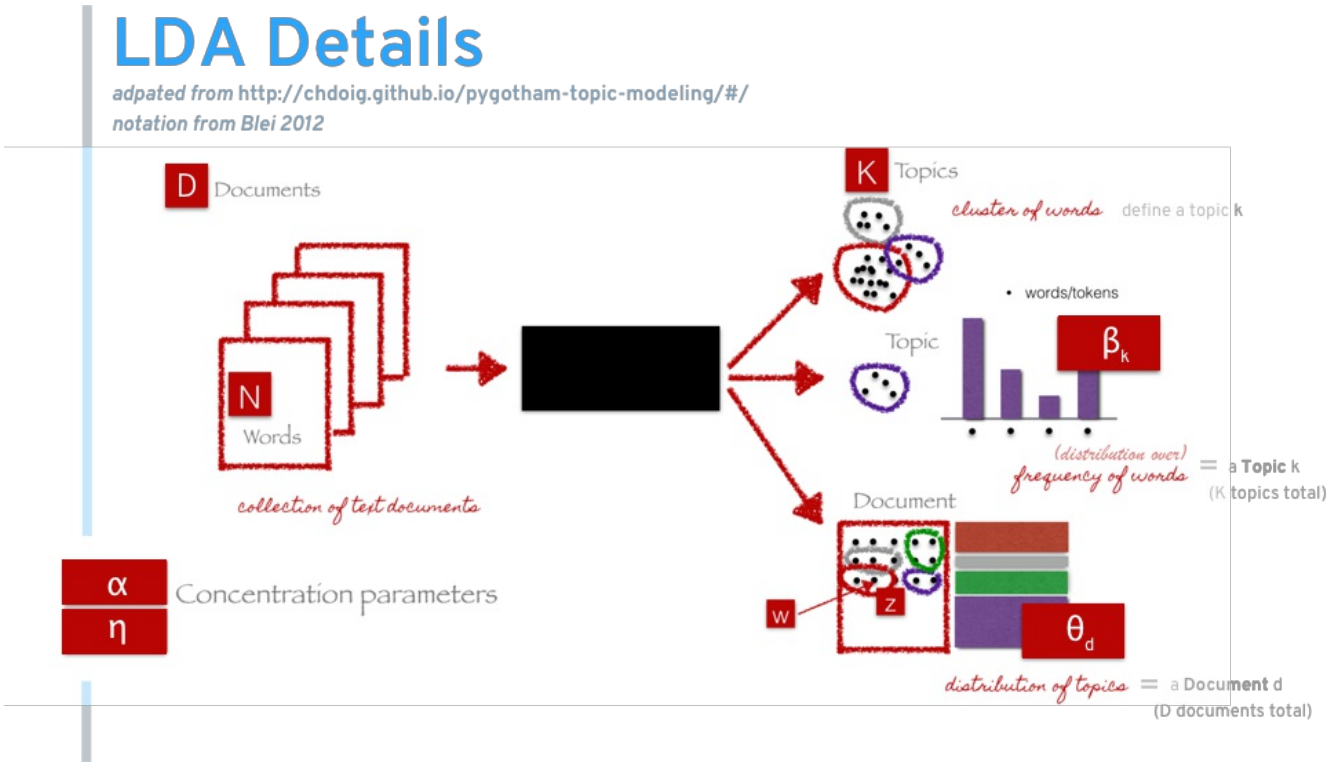
- **K topics**, where each topic is a collection (cluster) of words; note that most of the time K is prespecified by the user. This is where expert knowledge of a particular field can come in very handy
- **A topic-term matrix (`ttm`)**, a K x V matrix, which specifies a word distribution for each topic k; in other words, since each topic is a collection of words, it makes sense that some words are more "representative" of some topic k than other words. In this way, each topic has a distribution over words.
- **A document-topic matrix (`dtopm`)**, a D x K matrix, which specifies a topic distribution for each document d; in other words, since we can understand each document as having multiple topics, it makes sense that some topics are more saliently present in a document as its "main topics / main themes". Since this document-topic matrix is a vectorized representation of the doucments, it could also be used as an input to wordcloud, or as a feature for other machine learning algorithms.

6.4 The Details of LDA

For the reader interested in how LDA works, we present a more technical treatment of LDA here, which is heavily drawn from Blei 2012 and Christine Choig's Lecture Slides

Firstly, we note that the topic structure of our documents is *latent* (i.e.: hidden, unobserved). We *only* observe the documents and words themselves.

As such, LDA attempts to infer the hidden topic structure from the observed documents (Blei, 2012). Formally, LDA tries to compute the posterior distribution, or conditional distribution of the hidden structure given the observed documents. In Blei's words: "*what is the hidden structure that likely generated the observed collection?*"



To describe LDA formally, we introduce the following notation and (adpated) explanations from Blei 2012 to maintain consistency with standard notation:

- D, an fixed integer, representing the number of documents in our corpus
- V, a fixed integer, representing the vocaulary size, i.e.: the number of unique tokens in our corpus
- Topics $\beta_{1:K}$ (`K x 1` vector), where each `topic` β_k is a (discrete) distribution over the vocabulary. Recall that a `topic` k is a list of words, some of which are more "relavent" than others for the given topic k; in this way, each topic β_k is a discrete distribution over the vocabulary.
- Documents $\theta_{1:D}$, where document θ_d is a (discrete) topic distribution of the d-th document. $\theta_{d,k}$ (`real number` $\in [0, 1]$) represents the proportion of topic k in the d-th document. Recall that if we have K topics, each document is modelled as a mix of these K topics, each topic with some proportion between 0 and 1. This allows us to make statements such as "this document is 80 percent about time, 18 percent about genetics, and 2 percent about other micellaneous topics".
- The topic assignments for the d-th document are z_d(`n x 1 vector`), where $z_{d,n}$ (`integer` $\in [0, K]$) is the topic assignment for the nth word in document d; Recall that each document is thought of as a mixture of K topics, each with some proportion between 0 and 1. One way to get this proportion is by conceptualizing each word n in a document d as having a topic k, and the proportion of words "with" topic k is the proporition of topic k in document d
- The observed words for document d are w_d, where $w_{d,n}$ is the nth word in document d

The full joint probability distribution modelled by LDA is given by:

$$p(\beta_{1:K}, \theta_{1,D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) \left(\prod_{n=1}^{N} p(z_{d,n} | \theta_d) p(w_{d,n} | \beta_{1:K}, z_{d,n}) \right)$$

Recall that LDA infers the hidden topic structure by modelling the conditional distribution of the hidden structure given the observed documents. Formally, LDA is modelling:

$$\frac{p(\beta_{1:K}, \theta_{d,k}, z_{1:D}, w_{1:D})}{w_{1:D}}$$

$$p(\beta_{1:K}, \theta_{d,k}, z_{1:D} | w_{1:D}) =$$

The numerator, or joint probability distribution over all the random variables, is straightforward to compute, but the denominator, obtained by margininalizing over all latent topic structures, is exponentially large and intractible to compute (Blei 2012). As such, LDA is approximately computed in two main ways, through sampling-based algorithms and variational algorithms (Blei 2012). A sampling-based algorithm constructs an approximation of the posterior, and repeatedly draws samples from this approximation so in the limit converges to the true posterior distribution, and is non-deterministic (Blei, 2012). The most popular sampling-based algorithm is Gibbs Sampling. Variational methods are deterministic algorithms. They pre-suppose that the posterior distribution is from a parametrized family of distributions (e.g.: a bell curve that's centered or off center, narrowerer or flatter, depending on the parameters of mean and variance) (Blei, 2012). This makes variational methods an optimization problem as it tries to find the parameters that define a distribution closest to the posterior distribution. Here, closeness is measured by Kullback Liebler Divergence (KL-Divergence), which takes as input two distributions (order matters) and returns a real valued measure of closeness (Blei, 2012).

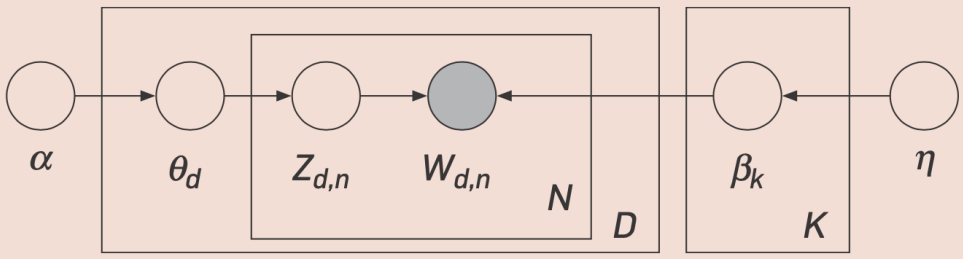
Assumptions of LDA (adapted from Blei, 2012):

- **Bag of words assumption**, that ordering of words in the document doesn't matter. We have seen two examples: term-document counts and tf-idf weighting.
- **Ordering of documents don't matter**; this is fairly realistic, unless we are modelling changes in topic over time. This can be an interesting area of further research in a followup study, where we can compare trip reports of the 1960's with those of the 21st century, to examine if and how the topics present in trip reports have evolved. Indeed, one can conceive that the 1960's may be marked by hippie culture, and extending the analysis through the dimension of time may reveal cultural around psychedelic substances. Blei 2012 features an example of dynamic LDA model, examining how the topics in the journal Science has evolved through time.
- **The number of topics K is assumed to be fixed**. To account for changing number of topics, one can turn to Bayesian non-parametric topic models, which is outside the scope of this thesis.

Topic Model Extensions, and Future Research Areas: See Wallach (2006) and Griffiths, Styvers & Blei (2015) for models that relax the bag of words assumptions use words conditioned on previous words and LDA in conjunction with a hidden markov model (HMM), which lead to improved performance. Other types of topic models include (Blei 2012): correlated topic model and pachinko allocation machine, which account for correlations between topics (e.g.: machine learning and statistics may be more correlated than machine learning and penguins; the spherical topic model allows words to be unlikely in a topic (for example, "coconuts" is unlikely in documents about watches); sparse topic models and “bursty” topic models respectively impose more constraints over topic distributions and more realistically model word counts (Blei 2012).

For those familiar with plate models, the following graphic (Blei 2012) reminds us that LDA is a type of a broader class of algorithms: probabilistic graphical models. D and N represent multiplicity, i.e.: the referenced cells are repeated D and N times. α and η are hyperparameters that specify the shape of two Dirichlet Distributions, used to describe the θ_d (the topic distribution of document d) distributions and β_k (word distribution of topic k) distributions respectively.

Figure 4. The graphical model for latent Dirichlet allocation. Each node is a random variable and is labeled according to its role in the generative process (see Figure 1). The hidden nodes—the topic proportions, assignments, and topics—are unshaded. The observed nodes—the words of the documents—are shaded. The rectangles are “plate” notation, which denotes replication. The *N* plate denotes the collection words within documents; the *D* plate denotes the collection of documents within the collection.



6.5 Implementation of LDA on psychedelic trip reports

In practice, LDA is implemented either from a Gibbs Sampling approach (e.g.: `mallet` package) or via Variational Methods (e.g.: `gensim`), and optimized using Expectation Maximization (EM) algorithms. We note without theoretical justification that Lyle Ungar has found empirically that `mallet` works better and provides more interpretable models (source: conversations). Luckily for us,

packages such as `mallet` and `sklearn` are easy to use, and provide state of the art LDA implementations. We first use `sklearn` implementation of LDA, which uses an online variational Bayes algorithm, due to its ease of use and compatibility with LDA visualization packages such as `pyLDAvis`.

In [26]:

```
# USE sklearn:
https://nbviewer.jupyter.org/github/bmabey/pyLDAvis/blob/master/notebooks/sklearn.ipynb

# imports for visualizing LDA
import pyLDAvis
import pyLDAvis.sklearn
# setting for jupyter notebooks
pyLDAvis.enable_notebook()

# inports for LDA and preparing data set
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation

####
# Prepare LDA inputs: vectorization of documents
# use the tfidf matrix we had before as the document vector represetation for input into LDA
# our lda input is simply dtm_tfidf, the document term matrix with tfidf weighting. We take a look
at the shape
print(dtm_tfidf.shape)

####
# Fit Latent Dirichlet Allocation models

# for TFIDF DTM
lda_tfidf = LatentDirichletAllocation(n_components=5, random_state=0)
lda_tfidf.fit(dtm_tfidf)
```

```
(19915, 107108)
```

Out[26]:

```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
                          evaluate_every=-1, learning_decay=0.7,
                          learning_method='batch', learning_offset=10.0,
                          max_doc_update_iter=100, max_iter=10,
                          mean_change_tol=0.001, n_components=5, n_jobs=None,
                          perp_tol=0.1, random_state=0, topic_word_prior=None,
                          total_samples=1000000.0, verbose=0)
```

In [47]:

```
# Fitting LDA with TF document term matrix
lda_tf = LatentDirichletAllocation(n_components=5, max_iter=5,
                                   learning_method='online',
                                   learning_offset=50.,
                                   random_state=0)
lda_tf.fit(dtm_tf)
```

Out[47]:

```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
                          evaluate_every=-1, learning_decay=0.7,
                          learning_method='online', learning_offset=50.0,
                          max_doc_update_iter=100, max_iter=5,
                          mean_change_tol=0.001, n_components=5, n_jobs=None,
                          perp_tol=0.1, random_state=0, topic_word_prior=None,
                          total_samples=1000000.0, verbose=0)
```

In [41]:

```
# print the top words for each topic

n_top_words = 20

def print_top_words(model, feature_names, n_top_words):
    print(model.components_.shape)
    print(len(feature_names))
    for topic_idx, topic in enumerate(model.components_):
```

```
    for topic_idx, topic in enumerate(model.components_):
        message = "Topic #%d: " % topic_idx
        message += " ".join([feature_names[i]
                             for i in topic.argsort()[:-n_top_words - 1:-1]])
        print(message)
    print()
```

```
(5, 107108)
107108
Topic #0: phenibut lyrica adrafinil vinpocetine dmae dhea pregabalin bdo selegiline aniracetam hyd
ergine ultracet mpa atenolol mucuna gvl olmifon tiletamine bup rosea
Topic #1: javier vassopressin pento hughie kirill tess rj sutts lingerie noss zworfin atterall mig
razone pmt coricedin burbon usel divinoram prulict empracet
Topic #2: like not time felt would one could experience feel trip back no feeling get really aroun
d started still effects much
Topic #3: ich und der zu das es mich auf die ein mir nicht mit war ist den eine auch sich noch
Topic #4: cp5 zam lagochilus zara mulungu slipper paan inebrians ebo cdt snapdragon brennan miguel
gannon tamas zippie rollladenschaft pvcs anita serequel
```

In [48]:

```
# Print the topics derived from tfidf document-term matrix
tfidf_feature_names = tfidf_vectorizer.get_feature_names()
print_top_words(lda_tfidf, tfidf_feature_names, n_top_words)
```

```
(5, 107108)
107108
Topic #0: phenibut lyrica adrafinil vinpocetine dmae dhea pregabalin bdo selegiline aniracetam hyd
ergine ultracet mpa atenolol mucuna gvl olmifon tiletamine bup rosea
Topic #1: javier vassopressin pento hughie kirill tess rj sutts lingerie noss zworfin atterall mig
razone pmt coricedin burbon usel divinoram prulict empracet
Topic #2: like not time felt would one could experience feel trip back no feeling get really aroun
d started still effects much
Topic #3: ich und der zu das es mich auf die ein mir nicht mit war ist den eine auch sich noch
Topic #4: cp5 zam lagochilus zara mulungu slipper paan inebrians ebo cdt snapdragon brennan miguel
gannon tamas zippie rollladenschaft pvcs anita serequel
```

In [49]:

```
# Print the topics derived from tf document-term matrix
tf_feature_names = tf_vectorizer.get_feature_names()
print_top_words(lda_tf, tf_feature_names, n_top_words)
```

```
(5, 107108)
107108
Topic #0: not like effects time feel feeling would much no day hours felt get one experience still
good drug first dose
Topic #1: dan non di la che un lance nita ce ci sarina mi come ed sarah per il sickie le sapo
Topic #2: like time felt not would back could one around trip started get really got went friend n
o still go thought
Topic #3: ich und die der es war das zu mich nicht mir auf mit ein den auch dass noch sich eine
Topic #4: experience not one mind no dmt body world life sense salvia reality state energy
psychedelic experiences space consciousness light many
```

In [1]:

```
# tt_matrix is the topic-term matrix, and gives us each topic (row) as a list of words (columns)
# Source: https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-meth
ods-for-text-data-f6f7d70acd41
# tt_matrix = lda_tfidf.components_
# for topic_weights in tt_matrix:
#     topic = [(token, weight) for token, weight in zip(vocab, topic_weights)]
#     topic = sorted(topic, key=lambda x: -x[1])
#     topic = [item for item in topic if item[1] > 5]
#     print(topic)
#     print()
```

The package pyLDAvis is the python implementation of the powerful R visualization package LDAvis for LDA model. The reader is

encouraged to play with package, which gives us interactive visualizations. It is commented out here due to its long run times.

In [454]:

```
###
# Visualization of Latent Dirichlet Allocation models
# using the tf-idf model to visualize TF-IDF
# pyLDAvis.sklearn.prepare(lda_tfidf, dtm_tfidf, tfidf_vectorizer)

# pyLDAvis.sklearn.prepare(lda_tf, dtm_tf, tf_vectorizer)
```

Results from tfidf DTM, with 5 topics

- Topic #0: phenibut lyrica adrafinil vinpocetine dmae dhea pregabalin bdo selegiline aniracetam hydergine ultracet mpa atenolol mucuna gvl olmifon tiletamine bup rosea
- Topic #1: javier vassopressin pento hughie kirill tess rj sutts lingerie noss zworfin atterall migrazone pmt coricedin burbon usel divinoram prulict empracet
- Topic #2: like not time felt would one could experience feel trip back no feeling get really around started still effects much
- Topic #3: ich und der zu das es mich auf die ein mir nicht mit war ist den eine auch sich noch
- Topic #4: cp5 zam lagochilus zara mulungu slipper paan inebrians ebo cdt snapdragon brennan miguel gannon tamas zippie rollladenschaft pvcs anita serequel

Results from tf DTM, with 5 topics

- Topic #0: not like effects time feel feeling would much no day hours felt get one experience still good drug first dose
- Topic #1: dan non di la che un lance nita ce ci sarina mi come ed sarah per il sickie le sapo
- Topic #2: like time felt not would back could one around trip started get really got went friend no still go thought
- Topic #3: ich und die der es war das zu mich nicht mir auf mit ein den auch dass noch sich eine
- Topic #4: experience not one mind no dmt body world life sense salvia reality state energy psychedelic experiences space consciousness light many

Firstly, we remark that the number 5 was completely arbitrary — finetuning the number of topics is in itself a non-trivial task, and provides a plethera of future research directions. Next, these topics selected are fasinating. First observing the topics extracted using the tfidf document-term matrix. Topic 0 seems to be drug names, many of which are nootropics (cognitive performance / mood enhancing drugs) and antidepressants. Topic 2 relates to time and experience, and seems to be an abstract expression of phenomenology. I cannot quite make sense of topics 1, 3, and 4, which seem to be in Italian and German. This was incredibly surprising for me as I had not realized there are such a significant portion of these foreign reports featured in the corpus.

The topics extracted using the term frequency document-term matrix, seem even more interesting. Topic 0 seems to deal primarilty with time, with the words "time", "day", "hours" and the experience of it ("experience", "feel", "like", "effects", "felt"). Topic 2 also mentions time, and seems to focus primarity on the logistics of trips, featuring simple words like "go", "around", "went", "still", "thought". Topics 1 and 3 are in Italian and German respectively. Topic 4 is by far the most interesting to me. It mentions "experience", "mind", "reality", "state", "energy", "space", "consciousness", all of which refer to the consciousnessness expanding, ineffability, and the viceral noetic qualities of the most powerful psychedelic experiences. Feeling at one with the universe, becoming present with the fabric of reality and noticing energy flows of space and time, are experiences that have been described both in literature (Stanislav Grof, 2009) and through conversations with colleagues and friends. Of particular note is the presence of both "dmt" and "salvia" in this topic — indeed, dmt is among the most powerful psychedelic substances in the world, and salvia remains one of the least understood and mysterious (See Rogan, 2018, Joe Rogan in conversation with Hamilton Morris for layman introductions). Together, dmt and salvia represent a certain frontier in our understanding of psychedelics.

7. Word Clouds: Getting the Gist

Word Clouds are an intuitively visual way to present vast amounts of textual information, revealing the key themes conveyed in a collection of documents. Prima facie, generating word clouds feel like a simple problem, yet very quickly deeper considerations arise, including (1) using the appropriate preprocessing, (2) whether to include documents of class $j \neq i$ when constructing a word cloud for documents of class, and (3) how to scale tokens, given (1) and (2).

7.1 Word Clouds using Term Frequencies

The most basic word clouds use term counts, or term frequencies, to scale the size of the words.

Now, we actually have two high levels choices regarding creating word clouds:

- a single word cloud represents a *single* trip report of a particular substance. This gives us report level wordclouds.
- a single word cloud represents *all* the reports of a particular substance. This gives us substance level wordclouds.

In the first approach, we can simply use the document-term matrix containing tfidf weightings we created earlier, which is of size D x V where D is total number of trip reports and V is the size of our vocabulary. In this representation, every trip report is represented as a vector. Hence, to get an idea what the experience of a particular substance is, we can randomly sample reports of a particular substance and create multiple word clouds for that substance.

In the second approach, we can concatenate all the reports for a particular substance into one document, for a total of 19 documents or `samples`. Under this schema, each substance is a single document, which is the concatenation of all the reports corresponding to that substance. It should be noted that that there are concerns with doing this, as `idf` typically assumes a large number of documents in the corpora.

Let's being with the first approach and then explore the second.

We will use our normalized corpus. Note that since each trip report may contain multiple classes, we will include a document that contains k classes in k word clouds.

7.1.1 Word Clouds for Single Trips with Term Frequencies

In [5]:

```
# # CHECKPOINT: Read in data if kernel has died
# import pandas as pd
# pd_trips_encoded_normalized = pd.read_csv("pd_trips_encoded_normalized.csv")
```

In [9]:

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

## Re create the Term Frequencies document term matrix as before
pd_trips_encoded_normalized.dropna(subset=['report'], inplace=True)
pd_trips_encoded_normalized["substance_unique_label"] =
pd_trips_encoded_normalized["substance_unique_label"].str.replace('.', '_')

norm_corpus = pd_trips_encoded_normalized["report"].to_list()

# the same procedure as above
tf_vectorizer = CountVectorizer(min_df=0., max_df=1.)
dtm_tf = tf_vectorizer.fit_transform(norm_corpus)
dtm_tf_nparr = dtm_tf.toarray()

vocab = tf_vectorizer.get_feature_names()
pd_report_vocab_tf = pd.DataFrame(np.round(dtm_tf_nparr, 2), columns=vocab)
pd_report_vocab_tf.head()
```

Out[9]:

	00	000	0000	00000	00000000	00000000000000000000000000003	00005	0001	0001g	0002	...	zzzs	zzzzang	zzzzz	zzzzzz	zzz
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

	00	000	0000	00000	00000000	000000000000000000000000000000	00005	0001	0001g	0002	...	zzzs	zzzzang	zzzzz	zzzzzz	zzz
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 107108 columns

In [10]:

```
# obtain the tf vectors of the reports for which we wish to create word clouds

from random import seed
from random import choice

# for ease of analysis, we set a seed so the random reports are the same every time
# change seed to generate wordclouds of differente reports
# also see: https://machinelearningmastery.com/how-to-generate-random-numbers-in-python/
chosen_seed = 14*(10**9)
seed(chosen_seed)

N_REPORTS_EACH_SUBSTANCE = 5 # number of reports to select for each substance
substance_reports_tf_map = {} # map {substance: {report: {vocab: tfidf weighting}}}

for substance in SUBSTANCES_OF_INTEREST_LIST:
    # list of indexes corresponding to reports only containing mushrooms
    index_substance_reports =
pd_trips_encoded_normalized.index[pd_trips_encoded_normalized['substance_unique_label'] ==
substance].tolist()
    # randomly select reports corresponding to each substance
    selected_report_indices = []
    for _ in range(N_REPORTS_EACH_SUBSTANCE): selected_report_indices.append(choice(index_substance
_reports))
    # Store tf vectors for each report
    substance_reports_tf_map[substance] = {}
    for reportindx in selected_report_indices:
        substance_reports_tf_map[substance][reportindx] = pd_report_vocab_tf.iloc[reportindx].to_di
ct()
```

In [14]:

```
###### PLOTTING WORDCLOUDS BASED ON REPORT LEVEL TF WEIGHTINGS ######
import matplotlib.pyplot as plt
from wordcloud import WordCloud, ImageColorGenerator


substances_to_plot = SUBSTANCES_PSYCHEDELICS_LIST + ["substance_mdma"]
num_substances = len(substances_to_plot)

# Plot the the various wordclodus
fig = plt.figure(figsize=(70, 60))
fig.suptitle("Word Clouds based on individual report TF weightings", fontsize=16)

print("Plotting word clouds with seed {} for substances: {}\n".format(chosen_seed,
substances_to_plot))
print("plotting {} rows (for number of substances) and {} columns (for num reports per substance)\
n\n".format(num_substances, N_REPORTS_EACH_SUBSTANCE))

inx = 0
for index_substance, substance in enumerate(substances_to_plot):

    for index_reportindx, reportindx in enumerate(substance_reports_tf_map[substance]):

        # WordCloud also takes a stopwords = [] as argument; omitted.
        wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white").generate_f
rom_frequencies(substance_reports_tf_map[substance][reportindx])
        # Subplot control: https://towardsdatascience.com/subplots-in-matplotlib-a-guide-and-tool-
for-planning-your-plots-7d63fa632857
#         print("plotting on ({}, {})".format(index_substance, index_reportindx))
        ax = plt.subplot2grid((num_substances, N_REPORTS_EACH_SUBSTANCE), (index_substance, index_r
eportindx)) # note here for subplots index begins at 0
        ax.set_title("{} Wordcloud".format(substance))
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        inx += 1

# plt.savefig('wordclouds_naive_psychedelics_tf.png')
```

```
# plt.savefig('wordclouds_naive_psychedelics_tf.png')
# plt.show()
```

```
Plotting word clouds with seed 14000000000 for substances: ['substance_mushrooms',
'substance_lsd', 'substance_mescaline', 'substance_5_meo_dmt', 'substance_dmt', 'substance_mdma']

plotting 6 rows (for number of substances) and 5 columns (for num reports per substance)
```

7.1.2 Word Clouds for Substances with Term Frequencies

We take slight pause to reflect upon our usage of the word cloud package. Here to create wordclous, we use the `wordcloud` package. Note that there are many tunable parameters in the package, and for this section, we are making numerous assumptions, including:

- Word scaling based on `frequency` only
- of words: the `wordcloud` package has a `relative_scaling` attribute, which specifies the importance of relative word frequencies for font-size. If `relative_scaling = 0`, the only word-ranks are considered; `relative_scaling=1` means words size is directly proportional to their frequency; Default of `relative_scaling = 0.5` considers both word frequency and word rank, and is used in our analysis. Insight retrieved from `wordcloud` documentation and from this tutorial

In [15]:

```
# Use the wordcloud package to generate baseline wordclouds
```

```
# Also see https://github.com/amueller/word_cloud
# Also see https://www.datacamp.com/community/tutorials/wordcloud-python

import numpy as np
import pandas as pd
from os import path
from PIL import Image
from wordcloud import WordCloud, ImageColorGenerator
from wordcloud import STOPWORDS as wordcloud_STOPWORDS
import nltk
from nltk.corpus import stopwords as nltk_stopwords
nltk.download('stopwords')

import matplotlib.pyplot as plt

pd_trips_encoded_normalized = pd.read_csv("pd_trips_encoded_normalized.csv")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/alextzhao/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [18]:

```
### Concatenate corpora for substances of interest

substances_all_list = pd_trips_encoded_normalized.loc[:,
'substance_mushrooms':'substance_syrian_rue'].columns.tolist()

# {substance_<name>: text for all reports containing substance}
substance_reports_map = {}

for substance in substances_all_list:
    # Concatenate with " " by default, TODO: Can specify separator as such: str.cat(sep="...")
    reports = pd_trips_encoded_normalized.query('{} == 1'.format(substance))["report"].str.cat(sep
= " ")
    substance_reports_map[substance] = reports
```

In [16]:

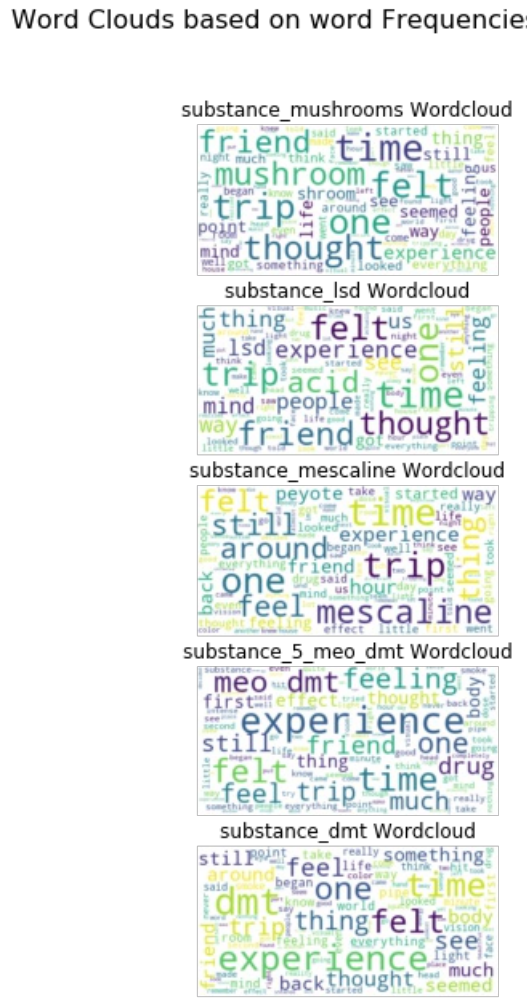
```
# ## Optionally perpare stopwords corresponding to substance names

# # a list of all the substances or
# substances_all_list = pd_trips_encoded_normalized.loc[:,
'substance_mushrooms':'substance_syrian_rue'].columns.tolist()
# # [ 'mushrooms', 'lsd', 'cannabis', ... etc] for use as stopwords
# substances_all_plainstring_list = [substance.replace("substance_", "") for substance in substanc
es_all_list]
# substances_all_plainstring_list = [substance.replace("_", " ") for substance in
substances_all_plainstring_list]

# substances_all_plainstring_list
```

In [19]:

```
### Create simple wordclouds based on term frequency

# a list of the "classical psychedelics"
substances_psychedelics_list = ["substance_mushrooms", "substance_lsd", "substance_mescaline",
"substance_5_meo_dmt", "substance_dmt"]

# NOTE: change this line of code to change which substances to plot
substances_to_plot = substances_psychedelics_list
num_plots = len(substances_to_plot)
print("Now plotting wordclouds for {} substances...".format(num_plots))

# creating subplots: https://stackoverflow.com/questions/25239933/how-to-add-title-to-subplots-in-
matplotlib
# changing figure size: https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-f
igures-drawn-with-matplotlib
fig = plt.figure(figsize=(50, 10))
fig.suptitle("Word Clouds based on word Frequencies", fontsize=16)

for index, substance in enumerate(substances_to_plot):
    # WordCloud also takes a stopwords = [] as argument; omitted.
```

```
    wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white").generate(subst
ance_reports_map[substance])
    ax = plt.subplot(num_plots, 1, index+1) # note here for subplots index must be [1, num_plots]
    ax.set_title("{} Wordcloud".format(substance))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")

plt.savefig('wordclouds_naive_psychedelics_tf.png')
plt.show()


### Plotting a single wordcloud
# wordcloud = WordCloud(stopwords=stopwords_all, max_font_size=50, max_words=100,
background_color="white").generate(reports_text_tokenized["substance_mushrooms"])
# plt.figure()
# plt.imshow(wordcloud, interpolation="bilinear")
# plt.axis("off")
# plt.show()
```

```
Now plotting wordclouds for 5 substances...
```



7.2 Wordclouds Using TFIDF Weightings

7.2.1 Word Clouds for Single Trips with TFIDF Weighting

In [21]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

## Re create the TFIDF weighted document term matrix as before
pd_trips_encoded_normalized.dropna(subset=['report'], inplace=True)
pd_trips_encoded_normalized["substance_unique_label"] =
pd_trips_encoded_normalized["substance_unique_label"].str.replace('.', '_')
```

```
norm_corpus = pd_trips_encoded_normalized["report"].to_list()

# the same procedure as above
tfidf_vectorizer = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
dtm_tfidf = tfidf_vectorizer.fit_transform(norm_corpus)
dtm_tfidf_nparr = dtm_tfidf.toarray()

vocab = tfidf_vectorizer.get_feature_names()
pd_report_vocab_tfidf = pd.DataFrame(np.round(dtm_tfidf_nparr, 2), columns=vocab)
pd_report_vocab_tfidf.head()
```

Out[21]:

	00	000	0000	00000	00000000	00000000000000000000000000003	00005	0001	0001g	0002	...	zzzs	zzzzang	zzzzz	zzzzzz	zz
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

5 rows × 107108 columns

In [23]:

```
# obtain the tfidf vectors of the reports for which we wish to create word clouds

from random import seed
from random import choice

# for ease of analysis, we set a seed so the random reports are the same every time
# change seed to generate wordclouds of differente reports
# also see: https://machinelearningmastery.com/how-to-generate-random-numbers-in-python/
chosen_seed = 2**17

N_REPORTS_EACH_SUBSTANCE = 5 # number of reports to select for each substance
substance_reports_tfidf_map = {} # map {substance: {report: {vocab: tfidf weighting}}}

for substance in SUBSTANCES_OF_INTEREST_LIST:
    # list of indexes corresponding to reports only containing mushrooms
    index_substance_reports =
pd_trips_encoded_normalized.index[pd_trips_encoded_normalized['substance_unique_label'] ==
substance].tolist()
    # randomly select reports corresponding to each substance
    selected_report_indices = []
    for _ in range(N_REPORTS_EACH_SUBSTANCE): selected_report_indices.append(choice(index_substance
_reports))
    # Store tfidf vectors for each report
    substance_reports_tfidf_map[substance] = {}
    for reportindx in selected_report_indices:
        substance_reports_tfidf_map[substance][reportindx] = pd_report_vocab_tfidf.iloc[reportindx]
.to_dict()
```

In [25]:

```
###### PLOTTING WORDCLOUDS BASED ON REPORT LEVEL TFIDF WEIGHTINGS ######

substances_to_plot = SUBSTANCES_PSYCHEDELICS_LIST + ["substance_mdma"]
num_substances = len(substances_to_plot)

# Plot the the various wordclodus
fig = plt.figure(figsize=(70, 60))
fig.suptitle("Word Clouds based on individual report TFIDF weightings", fontsize=16)

print("Plotting word clouds with seed {} for substances: {}\n".format(chosen_seed,
substances_to_plot))
print("plotting {} rows (for number of substances) and {} columns (for num reports per substance)\
n\n".format(num_substances, N_REPORTS_EACH_SUBSTANCE))

inx = 0
for index_substance, substance in enumerate(substances_to_plot):
```

```
    for index_reportindx, reportindx in enumerate(substance_reports_tfidf_map[substance]):

        # WordCloud also takes a stopwords = [] as argument; omitted.
        wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white").generate_f
rom_frequencies(substance_reports_tfidf_map[substance][reportindx])
        # Subplot control: https://towardsdatascience.com/subplots-in-matplotlib-a-guide-and-tool-
for-planning-your-plots-7d63fa632857
#         print("plotting on ({}, {})".format(index_substance, index_reportindx))
        ax = plt.subplot2grid((num_substances, N_REPORTS_EACH_SUBSTANCE), (index_substance, index_r
eportindx)) # note here for subplots index begins at 0
        ax.set_title("{} Wordcloud".format(substance))
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        inx += 1

# plt.savefig('wordclouds_naive_psychedelics_tf.png')
# plt.show()
```

```
Plotting word clouds with seed 131072 for substances: ['substance_mushrooms', 'substance_lsd', 'su
bstance_mescaline', 'substance_5_meo_dmt', 'substance_dmt', 'substance_mdma']

plotting 6 rows (for number of substances) and 5 columns (for num reports per substance)
```

7.2.2 Word Clouds for Substances with TFIDF Weighting

In [22]:

```
## See TFidfVectorizer docs: https://scikit-
learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse.csr import csr_matrix # need this if we want to save tfidf_matrix; see
https://stackoverflow.com/questions/34449127/sklearn-tfidf-transformer-how-to-get-tf-idf-values-of
-given-words-in-documen


# NOTE: reports_text_tokenized stores the tokenized corpus
# List of substances under consideration, sorted to preserve alphabetical ordering
substances_list_sorted = sorted(list(substance_reports_map.keys()))
substances_list_sorted

# TfidfVectorizer expects a list of strings, so we prepare the corpus in the appropriate format
corpus_psychedelics_sorted = []
for substance in substances_list_sorted:
    corpus_psychedelics_sorted.append(substance_reports_map[substance])
```

Note that by using TFidfVectorizer, we are also making the following assumptions. The following outlines the parameters of TfidfVectorizer for reference; additional information found in documentation

- `strip_accents=ascii` helps us strip accented characters such as `á` to `a`
- `lowercase=True` : automatically converts all the text to lowercase
- `analyzer=word` : considers words as the basic unigram unit; we can also do a character level model with `char`
- `stop_words=english` : uses built in stopwords given by `sklearn` ; Follow these instructions to view all the stopwords for `sklearn` and `nltk`
- `ngram_range=(1, 1)` : setting to (1, 1) gives us unigrams. For bigrams and trigrams, we use (1, 2) and (1, 3) respectively
- `max_df=1.0` : using the float `1.0` sets the maximum document frequency to 100\% of the documents
- `min_df=1` : ignores all words with document frequency less than 1
- `max_features=None` : Don't put a cap on the number of features
- `binary=False` : Use the tf counts, instead of setting all non-zero counts to 1
- `dtype=float64` : Default datatype for tf-idf values
- `norm=l2` : uses the L2 norm for each row of tf-idf values, so each row sums to 1
- `use_idf=True` : use idf weighting (instead of just tf)
- `smooth_idf=True` : this is akin to laplace smoothing, and helps us prevent 0 divisions
- `sublinear_tf=False` : Do not replace tf with 1 + log(tf)

In [28]:

```
# create and use a TfidfVectorizer
# We specify explicitly the parameters used, even for defaults, in order to be precise with our as
sumptions
# and for reproducibity
# note: sklearn by default throws away punctuations; we accept this as we are not performing senti
ment analysis
# and related tasks where punctuations may be highly informative

# https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/#.XeLZRzJKjmE
vectorizer_wc_tfidf = TfidfVectorizer(strip_accents='ascii',
                                      lowercase=True,
                                      analyzer='word',
                                      stop_words='english',
                                      ngram_range=(1, 1),
                                      max_df=1.0,
                                      min_df=1,
                                      max_features=None,
                                      vocabulary=None,
                                      binary=False,
                                      dtype='float64',
                                      norm='l2',
                                      use_idf=True,
                                      smooth_idf=True,
                                      sublinear_tf=False)

X_vectors_tfidf = vectorizer_wc_tfidf.fit_transform(corpus_psychedelics_sorted)
# vectorizer_tfidf.get_feature_names()
```

```
/Users/alexthao/anaconda3/envs/webmarkov/lib/python3.7/site
```

```
/Users/alextznao/anaconda3/envs/wokemonkey/lib/python3.7/site-
packages/sklearn/feature_extraction/text.py:1616: UserWarning: Only (<class 'numpy.float64'>,
<class 'numpy.float32'>, <class 'numpy.float16'>) 'dtype' should be used. float64 'dtype' will be
converted to np.float64.
  UserWarning)
```

In [247]:

```
# Take a look again at the sorted ordering of the substances
# substances_list_sorted
```

In [29]:

```
# convert sparse tfidf matrix to np array
substance_term_matrix_tfidf_nparr = X_vectors_tfidf.toarray()
vocab = vectorizer_wc_tfidf.get_feature_names()

# create dataframe mapping substance to vocabulary
pd_substance_vocab_tfidf = pd.DataFrame(np.round(substance_term_matrix_tfidf_nparr, 2),
columns=vocab, index = substances_list_sorted)
# pd_substance_vocab_tfidf.head()
```

In [31]:

```
# Create vocab: tfidf weighting maps for each substance, so we can use them in wordclouds

# map from {substance_<> : {vocab: tfidf weighting}}
substance_tfidf_map = {}

for substance in substances_list_sorted:
    substance_tfidf_map[substance] = pd_substance_vocab_tfidf.loc[substance].to_dict()
```

In [32]:

```
### Create simple wordclouds based on TFIDF weights

# a list of the "classical psychedelics"
substances_psychedelics_list = ["substance_mushrooms", "substance_lsd", "substance_mescaline",
"substance_5_meo_dmt", "substance_dmt"]

# NOTE: change this line of code to change which substances to plot
substances_to_plot = substances_psychedelics_list
num_plots = len(substances_to_plot)
print("Now plotting wordclouds for {} substances based on TFIDF...".format(num_plots))

# creating subplots: https://stackoverflow.com/questions/25239933/how-to-add-title-to-subplots-in-
matplotlib
# changing figure size: https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-f
igures-drawn-with-matplotlib
fig = plt.figure(figsize=(50, 10))
fig.suptitle("Word Clouds based on substance TFIDF weightings", fontsize=16)



for index, substance in enumerate(substances_to_plot):
    # WordCloud also takes a stopwords = [] as argument; omitted.
    wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white").generate_from_
frequencies(substance_tfidf_map[substance])
    ax = plt.subplot(num_plots, 1, index+1) # note here for subplots index must be [1, num_plots]
    ax.set_title("{} Wordcloud".format(substance))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")

plt.savefig('wordclouds_naive_psychedelics_tf.png')
plt.show()


### Plotting a single wordcloud
# wordcloud = WordCloud(stopwords=stopwords_all, max_font_size=50, max_words=100,
background_color="white").generate(reports_text_tokenized["substance_mushrooms"])
# plt.figure()
# plt.imshow(wordcloud, interpolation="bilinear")
# plt.axis("off")
# plt.show()
```

```
# plt.show()
```

```
Now plotting wordclouds for 5 substances based on TFIDF...
```

7.3 Word Clouds with Other Feature Vectors

Recall in earlier discussions that as long as we can represent documents as vectors, we can use those vector reprsentations for word clouds, clustering, and other tasks. Wordclouds require some measure of *importance* of each word to a document — in other words, as long as we have a {word: real number} mapping for each document, we can construct word clouds. With this foundation, we can in principle use the outputs of other algorithms to construct word clouds. For example, we can use the following:

- **Logistic regression weights**: classify documents by their substance label, and use the logistic regression weights for the word features as a measure of importance for particular words to predicting substance label. These weights (after the appropriate centering, scaling, and whitening), can be used as a measure of how important certain words to particular substances
- **Weights from other classification methods**: the above can be generalized to other classification methods such as decision trees, in which metrics such as information gain can be used.
- **LDA document embeddings**: Recall that LDA gives us (1) K topics, (2) a topic-term matrix, and (3) a document-topic matrix. In particular, the document-topic matrix gives us vector representations of each document, where each element of the vector is an integer representing the key of a topic. We can lookup the words best representing each topic using the topic-term matrix, and therefore compute the most representative words for each document. Using this as a baseline, we can then construct both document level and substance level wordclouds using the outputs from topic models.

`Tag: Future Directions`

7.4 Log Odds Ratio, and Homage to Statistical Inference

Previously we have discussed a variety of approaches based on using document vectorizations as starting points for generating wordclouds. While we will not go into the implementation, it is worth noting another possible approach based on work done by Chris Callison-Bruch.

This approach uses the Informative Prior Log Odds IPLO ratio method, which is used in natural language processing to identify the

differences in language usage patterns between two groups. The foundations rests upon the log odds ratio, a widely used technique in statistical inference and hypothesis testing. Intuitively, we ask, how much more likely is a word to belong to a particular document, as opposed to in another document.

For those interested, the theoretical framework is outlined in more detail here, based on work done in Text Simplification.

In general, IPLO is designed to compare two groups (aka, documents with binary labels). In our case of multilabelled psychedelic trip reports, we can take a one-vs-all approach, i.e.: compare documents with the label `substance.mushrooms` and documents without the label `substance.mushrooms`. This will discover words / tokens that are particularly informative of the `substance.mushrooms` class. Note again that reports with a particular `substance.*` label are not necessarily uniquely `substance.*`, and may contain other labels since people have a tendency to consume multiple different types of drugs simultaneously.

For those interested, one can use the code from john-hewitt and Chris Callison Burch's research group, which can be obtained by emailing Chris Callison Burch Directly.

`Tag: Future Directions`

7.5 (For Fun) Creating custom word cloud masks

To make the wordclouds more interesting, we can add custom image masks to the wordclouds, as inspired by this article, this tutorial, and `wordcloud` package documentation. Here, we show a wordcloud of reports related to mushrooms in the shape of a mushroom. It is fascinating the top two words are "time" and "one"

In [397]:

```
## Plot wordcloud with image mask
# Make sure white parts are 255 instead of 0, as per:
https://www.datacamp.com/community/tutorials/wordcloud-python
# TODO: Why is the wordcloud blurry, and why can I not get the mask to work appropriately?
# def transform_format(val):
#     if val == 0: return 255
#     else: return val

mask_mushroom = np.array(Image.open("images/util_mushroom_mask_color.png"))
# vectorization of transform_format function above; invert the mask to be in the right format

# Transform your mask into a new one that will work with the function:
# transformed_mask_mushroom = np.ndarray((mask_mushroom.shape[0], mask_mushroom.shape[1]), np.int3
2)

# for i in range(len(mask_mushroom)):
#     transformed_mask_mushroom[i] = list(transform_format(mask_mushroom[i]))

mask_mushroom[mask_mushroom != 0] = 1
mask_mushroom[mask_mushroom == 0] = 255
# mask_mushroom

#plot a mushroom wordmap about mushrooms
#increase "resolution": https://stackoverflow.com/questions/28786534/increase-resolution-with-word
-cloud-and-remove-empty-border
wordcloud_mushroom_masked = WordCloud(width=1000, height=10000, mask=mask_mushroom, max_font_size=5
0, max_words=100, background_color="black").generate(substance_reports_map["substance_mushrooms"])
# wordcloud_mushroom_masked = WordCloud(max_font_size=50, max_words=100, mask=mask_mushroom, backg
round_color="white").generate(substance_reports_map["substance_mushrooms"])

plt.figure(figsize=(20, 20))
plt.imshow(wordcloud_mushroom_masked, interpolation="bilinear")
plt.axis("off")

plt.savefig('wordcloud_naive_mushroom_masked')
plt.show()
```

7.6 WordClouds Appendix: Some Interesting Observations from `Tfidfvectorizer` output

It's interesting to note that looking at the Tfidfvectorizer feature_names, we see many numbers. These numbers either stand alone or are followed by units and quantifiers.

These units seem to be used to specify multiple types of quantities, such as:

- weight (that of substance ingested or user):
  - 'c', 'g', 'gm', 'gram', 'grams', 'lbs', 'mcg', 'mg', 'microgram', 'mics', 'o', 'oz', 'pounds', 'ug'
- volume or length
  - 'cc', 'cm', 'cup', 'ft', 'i', 'inch', 'iu', 'kg', 'kgs', 'meters', 'metres', 'mile', 'miles', 'ml', 'sq', 'tbs', 'tbsp', 'yards'
- time or time period
  - "'s' (e.g.: in 'the 1990s')", 'XXhXXm', 'am', 'hours', 'hr', 'min', 'minute', 'pm', 'sec'
- user age
  - 'day', 'y', 'yo', 'yr'
- scales and dimensions
  - "'5x' (e.g.: 5x salvia extract)", '1x15', '1x50mg'
- addresses
  - "st"
- speed
  - "mph"
- numerical quantification
  - "'strips' (e.g.: number of lsd strips)", "'ths' (as in 100ths times)", 'nd', 'th'
- sound information
  - 'bpm', 'bps', 'hz'

A closer examination of this numberical data may yield very interesting information about user demographics, set and setting, and dosage. All of this may be extremely valuable since dosage, set and setting, are among the most important factors in determining one's subjective experience when consuming a drug, and demographics may be correlated with the valence of a particular experience.

8. Visualizing High Dimensions: Clustering and Principle Component Analysis (PCA)

`Tag: Future Directions`

8.1 High Dimensionality: Everpresent and Difficult to Grok

Returning to the discussion of how vectorizing documents, we note explictly that using bag of word methods such as term frequencies and tfidf weightings, each document is represented by a vector of size V, most of which is filled with zeros. This vector representation is extremely sparse and high dimensional (V dimensions)! Even dense word and document embeddings are often on the order of few hundred dimensions, which is also very high dimensional. Indeed, the concept of high dimensionality is not just specific to language; other data sources such as images (a 28 by 28 pixel black and white photograph is $28^2 = 784$ dimension), videos, sound, fMRI data etc. are also high dimensional. Indeed, a substantial class of data in machine learning is high dimensional.

Why is high dimensional data interesting? Firstly, it is extremely hard to conceptualize, and essentially impossible to visualize the unprocessed data (how do you represent 784 dimensions on a 2D screen?). Notions of distances in high dimensions also become very non-intuitive, and it's difficult to fully appreciate how "vast" a 784 dimensional space truly is. The performance of algorithms also depend on dimensionality, with algorithms such as linear and logistic regression failing (without regularization) when number of features exceed number of observations, and the performance of various algorithms increasing first then decreasing with added dimensionality (Hughes Phenomenon). Because high dimensionality is so everpresent, the phenomena that arises only when analyzing high dimensional data is often called the "Curse of Dimensionality" (Bellman 1964)

8.2 Clustering

"Clustering" refers to a broad class of unsupervise techniques that group observations together by some similarity metric. For example, we can cluster customers into different segments based on their spending habits or demographics, we can cluster images of brains based on particular features detected by an fMRI or we can cluster psychedelic trip reports they are feature vectors. As long as we have a vector representation of our observations, a distance metric such as the normal Euclidean distance (L2 Norm), we can use a variety algorithms (such as Agglomerative Hiearchical Clustering, K Means Clustering etc.) to cluster psychedelic trip reports together.

Clustering based on LDA features

As a simple example, we can cluster trip reports together based on their LDA features, in particular by using the document-topic matrix. This code has been commented out, but the reader is welcome to adapt and modify to examine the behavior of clustering algorithms.

In [33]:

```
# Fit LDA model to obtain document-topic matrix
# Adapted from: https://towardsdatascience.com/understanding-feature-engineering-part-3-
traditional-methods-for-text-data-f6f7d70acd41
```

```
traditional-methods-for-text-data-1017d70ac41
# from sklearn.cluster import KMeans
# from sklearn.decomposition import LatentDirichletAllocation

# n_components = 3

# print("Fitting LDA model with {} components using TFIDF vectorization of
documents...".format(n_components))
# lda = LatentDirichletAllocation(n_components=3, max_iter=10000, random_state=0)

# # document topic matrix
# print("Creating document-topic matrix...")
# dtopm = lda.fit_transform(dtm_tfidf)

# print("Rendering data frame with document-topic features")
# dtopm_features = pd.DataFrame(dtopm, columns=['T1', 'T2', 'T3'])
# dtopm_features.head()
```

In [ ]:

```
# # due to long run time of LDA model, save to disk
# import pickle

# with open("dtopm_features_tfidf_3components", "wb") as out_file:
#     pickle.dump(dtopm_features, out_file)
```

In [ ]:

```
# # perform k means clustering with LDA features
# print("Now performing K-means clustering with LDA document-topic features...")
# km = KMeans(n_clusters=5, random_state=0)
# km.fit_transform(features)

# print("Finished clustering, now rendering data frame")
# cluster_labels = km.labels_
# cluster_labels = pd.DataFrame(cluster_labels, columns=['cluster_label'])
# pd_trips_encoded_normalized_clusters = pd.concat([pd_trips_encoded_normalized, cluster_labels],
axis=1)

# print("Writing df with cluster labels to disk...")
# pd_trips_encoded_normalized_clusters.to_csv("pd_trips_encoded_normalized_clusters.csv", index =
False)
```

In [ ]:

```
# %%R


# R_trips_encoded_normalized_clusters <- read.csv("pd_trips_encoded_normalized_clusters.csv")
# glimpse(R_trips_encoded_normalized_clusters)

# R_trips_encoded_normalized_clusters.long <-
R_trips_encoded_normalized_clusters.gather(substance_key, isPresent, substance_lsd:substance_UNK)

# # examine the substance distribution for each cluster
# R_trips_encoded_normalized_clusters.long %>%
#     filter(isPresent == 1) %>%
#     group_by(cluster_label, substance_key) %>%
#     summarise(count = n()) %>%
#     mutate(percent = round(count / sum(count) * 100, digist=1)) %>%
#     arrage(desc(count))


# # TODO: Examine the top words associated with each cluster

# #### R template
# # R_trips_encoded_normalized_eda %>% filter(substance_lsd == 1) %>%
# #     select(-substance_lsd) %>%
# #     gather(substance_key, isPresent, substance_mushrooms:substance_UNK) %>%
# #     filter(isPresent == 1) %>%
# #     group_by(substance_key) %>%
# #     summarise(count = n()) %>%
# #     mutate(percent = round(count / sum(count) * 100, digits = 1)) %>%
# #     arrange(desc(count))
```

8.3 Principle Component Analysis

`Tag: Future Directions`

PCA Theory

Principle Component Analysis (PCA) is an unsuperived technique whose chief purpose is to reduce the dimensionality of data, while sacrifing as little information as possible from the original dataset. When the data is reduced to one, two, or three dimensions, PCA has the added benefit of serving as a data visualization tool. Since our data is basically a set of vectors which span a vector space, the notion of "direction" in this vector space is well defined. Furthermore,just as data with only x and y values vary in either the x or y direction, or really along any angle (evident when plotted on a scatter plot), a data in (say) 100,000 dimensions also vary in each of those 100,000 directions, as well as all the directions in between. It turns out "reduce dimensionality while sacrficing as little information as possible" is equivalent to finding the directions in which the data varies the most. That is, if we were reducing our data to three dimensions, we need to find the three directions in the 3D space in which the data varies the most. These directions are called `principle component loading vectors` (James et. al, 2013).

Recall a direction can be expressed as a vector, which has magnitude and direction, Let our data be an nxp matrix (n observations, each with p featuers). Consider the linear combination of our original features X_j:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$$

The vector $\phi_1 = \{\phi_{11}, \phi_{21}, \cdots, \phi_{p1}\}$ is typically called the **first principle component loading vector**, a p-dimensional unit vector which precisely defines the direction in our original feature space along which the data varies the most. The elements of ϕ_1 are often called the *loadings* of the first principle component.

With this direction specified, we still need a way to represent all of the n points in our dataset. Indeed, we can project all the points (each of which are p dimensional vectors) onto ϕ_1 by taking the dot product of each observation's p-dimensional vector and ϕ_1. Since the dot product is a scalar, each point in our original dataset has been reduced to a single dimension, for a total of n values, one for each of our original data points. These n values, $\{z_{11}, z_{21}, \cdots, z_{n1}\} = z_1$ are called the *scores* of the first principle component. In this way, our p-dimensional dataset (n x p) has been dimensionality reduced to a single dimension (n x 1).

Now, PCA is not limited to reducing data down to a single dimension. We can reduce our data to M dimensions by finding the first M principle component loading vectors $\phi_1, \phi_2, \cdots, \phi_M$, each of which are p dimensional vectors (James et. al, 2013). The first principle component loading vector, ϕ_1 is the direction along which the data varies the most. The second principle component loading vector, ϕ_2, is the direction orthogonal to ϕ_1 along which the data varies the most. ϕ_3 is the direction orthogonal to ϕ_1 and ϕ_2 along which data varies the most, so on and so forth, with each subsequent loading vector orthgonal to all previous ones and attempting the explain the most variance. (This orthogonality requirement is due to principle component key to PCA's optimality garuntees, and relates to Singular Value Decomposition,, which is outside the scope of this thesis; readers interested are invited to review James et. al, 2013.

With these M principle component loading vectors, to reduce high dimensional data to M dimensions, we project each of our n data points onto each of the M principle component loading vectors, such that each data point is now represented by an M dimensional vector. The projected values onto the m-th principle component loading vector are called the scores of the m-th principle component (n x 1 vector).

Precisely, we refer to the linear combination $Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$ with the highest variance as the first *principle component* of the set of features $X_1, X_2, \cdots, X_p$. PCA then finds the user-specified M (orthonal) principle components that best describe the data, which minimizes reconstruction error (i.e.: sacrificing as little information as possible) and maximizes total variance explained (finding the loading vectors corresponding to the directions in which the data varies the most).

PCA Uses

We mention without demonstration that PCA can be used as another visualization tool to better understand our corpus of psychedelic trip reports. And more generally, this idea of projecting dataset onto a lower dimensional subspace spanned by some number of basis vectors (which could be PCA loading vectors, or any set of arbitrary basis vectors). One application is to visualize document vector embeddings by projecting them onto the first and second principle components. We can also visualize word embeddings (trained / fine tuned using our data set) using similar procedures. The principle component scores of each observation calcualated by PCA (at both the document and word level) can also be used as input feature vectors for other algorithms such as clustering, classification tasks. Due to the scope of the thesis, we will not dive too deeply into PCA, and rather earmark this technique as a possible future direciton.

9: Predicting Substance Labels from Trip Reports

`Tag: Future Directions`

In this section, we describe possible additional analysis using our corpus by exploring the classification problem, without going into in depth detail. The reader is invited to skip this section.

9.1 The Classification Problem: Our Goals

We have a corpus of psychedelic trip reports, where each report is labelled by discrete substance labels. As such, we can attempt to predict the substance(s) consumed given a partiular trip report. This opens up a whole world of possible techniques and feature engineering considerations that can be a topic of a separate thesis.

9.2 Multilabel Considerations

Typically a dataset prepared for classification has one discrete label per observation. There may be only two possible classes (e.g: sick vs. not sick; binary classification problem) or multiple possible classes (multi-class classification problem). It is fairly rare to have multiple discrete labels per observation. When there are more than two possible classes, we have a **multilabel multiclass classification task** (also called Multioutput-multiclass classification or multi-task classification). This is a fascinating problem that is outside the scope of the thesis; interested readers can check out algorithms from sklearn that handle multiclass and multilabel learning, which are generalizations of Trees, KMeans, and Random Forest.

We can also pre-process our substance labels to find the most co-consumed substances and make substance pairs their own labels. This would greatly expand the number of classes, but would reduce the multilabel multiclass classification task into just a multiclass classification task which is more well studied. These two formulations would be asking two different questions, one about whether a substance is *present* and the other treating co-consumed substance as a unitary experience. Also see multilabel classification examples from sklearn

9.3 The Power of Transfer Learning: State of the Art

`Tag: Future Directions`

Beyond traditional classification techniques such as those featured in sklearn, a very interesting direction for future research is to fine tune state of the art language models such as BERT and GPT-2 using a psychedelics trip report corpus for a classification task. Huggingface offers these transformer models such as BERT for fine tuning.

9.5 NLP in Practice: Using automatic piplines for text feature extraction and classification

`Tag: Future Directions`

Speaking with friends and professors in the industry, it became apparent that one of the most time consuming aspects of machine learning and more specifically natural language processing in practice is selecting the appropriate algorithms, and then the hyperparameters for the models. For example, sklearn has more than 10 different classifers, each more suitable for various kinds of input data. Once a classification algorithm is selected, the hyperparameters related to both model definition (e.g.: model size, tree depth etc.) and learning (e.g.: number of iterations, learning rate etc.) still has to be tuned.

With these considerations, a compelling future direction is to dive more deeply into automatic NLP pipelines in production

environments, to better understand the various feature extraction and hyperparameter tuning steps (e.g.: gridsearch?) A fairly recent release (in 2015) is AutoML, a framework leveraging scikit learn and automates hyperparameter selection. AutoML features "15 classifiers, 14 feature preprocessing methods, 4 data preprocessing methods, giving rise to a structured hypothesis space with 110 hyperparameters" (Feurer et. al, 2015). AutoML features *warmstarting* (uses hyperparemteres that worked well previously for similar problems), *bayesian optimization* (uses random forest to predict optimal hyperparameters), and *ensembles* the 50 best classifiers (Feurer et. al, 2015). The interest user is invited to visit AutoML homepage and also check out an example of a feature extraction pipline from sklearn.

10. Generation of Trip Reports

`Tag: Future Directions`

Another fascinating problem domain is that of text generation. What would it look like for the computer to automatically write trip reports?

A suitable problem formulation that corresponds to this problem is: "given a sequence of p characters (or words), predict the next character (or word)". The exact formulation gives rise to character level or word level models, each of which can also be unigram, bigram, trigram, or in general n-gram based. Readers interested in this problem domain are encouraged to research "Hidden Markov Models (HMM)", "Long Short Term Memory Networks" (LSTM's, see here for an excellent blog post by Chris Olah), "BERT" (from Google) and "GPT-2"(from OpenAI).

For a an example of generating Drake Lyrics with HMM's, Decision Trees, and LSTM's, see here.

11. Feature Importance

`Tag: Future Directions`

One of the main concerns with machine learning is its black box nature. Especially as neural networks get deeper with current state

of the art models (e.g.: Google's models with well over 1 billion parameters), it becomes extremely difficult to demystify the inner workings of these complex models, and understand sources of performance, bias, and shortcomings. If we are to better understand how language conveys our inner experiences, and use trip reports robustly in psychiatry, psychology, and neuropharmacology research (as well as developing appropriate neurotargets for neurotechnology), fdemystifying our machine learning algorithms become critically important.

11.1 LIME: State of the Art Feature Importance

LIME stands for "Local Interpretable Model-Agnostic Explanations", a technique developed by Marco Tulio Ribeiro, , Sameer Singh and Carlos Guestrin at the University of Washington for demystifying machine learning algorithms (Ribeiro, Singh & Guestrin, 2016). Taking a trained model and a particular observation as input, the process highlights the features of the observation that most informed the model's "decision making process".

Two examples from Ribeiro et. al's blog post makes this clear:

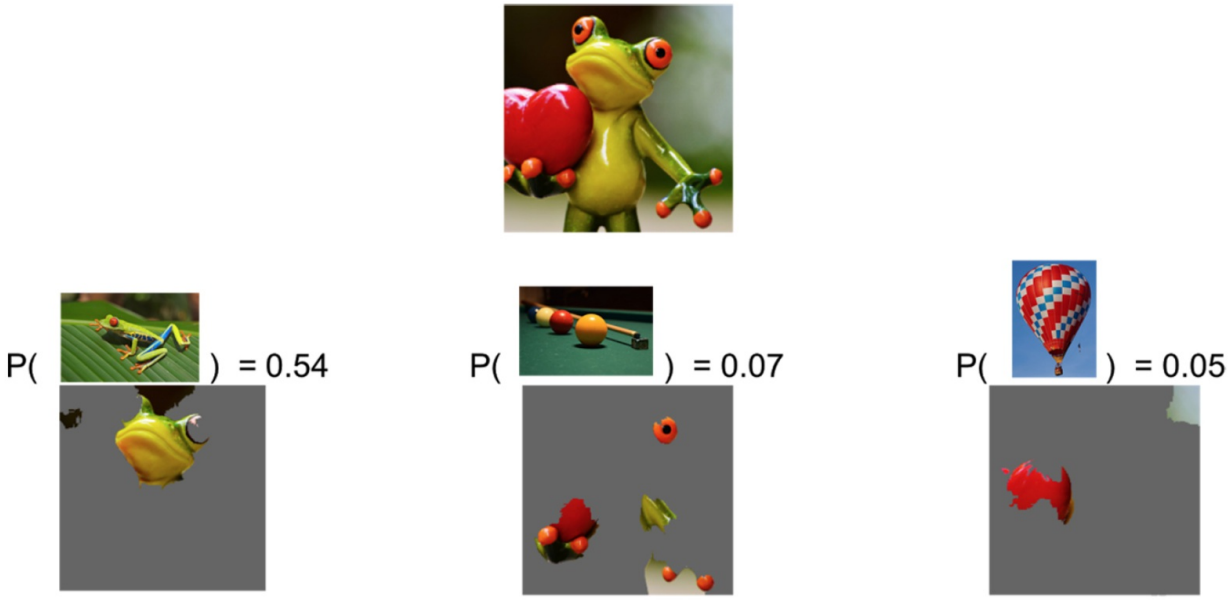


Figure 6. Explanation for a prediction from Inception. The top three predicted classes are "tree frog," "pool table," and "balloon." Sources: Marco Tulio Ribeiro, Pixabay (frog, billiards, hot air balloon).

In this example of the tree frog, the top three classe predicted by the trained model (Google's Inception Model) of the given input image were `tree frog`, `billiards`, and `hot air balloon`. We see that LIME is able to highlight the portions of the input image most informed the model's predictions.

LIME can also be used for textual data, as seen by the following example (also from Ribeir et. al's blog post) of a random forest classifer predicting whether an email is `Christian` or `athiest`. The classifer achieved an accuracy of 92.4%, but we see through LIME it performed so well entirely for the wrong reasons (Ribeiro, Singh & Guestrin, 2016).

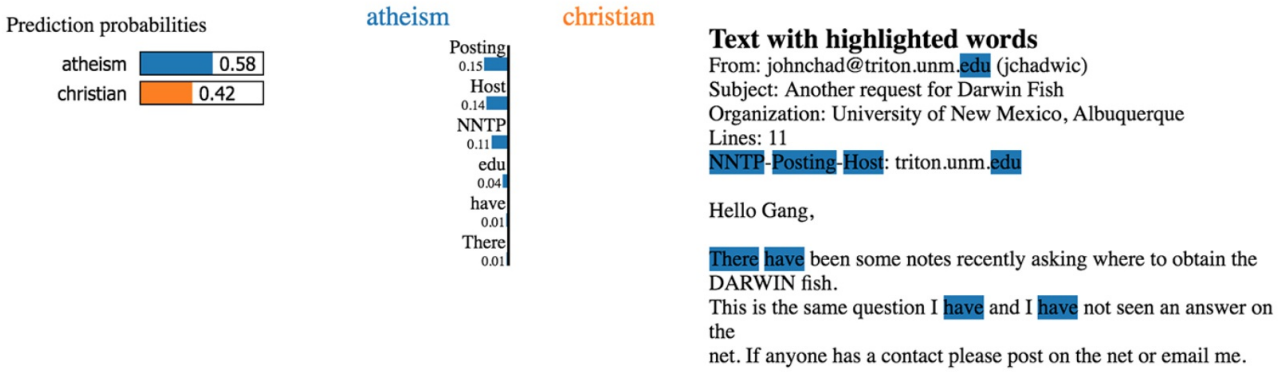


Figure 5. Explanation for a prediction in the 20 newsgroups data set. Source: Marco Tulio Ribeiro.

Why is this important? As medicine increasingly integrates machine learning processes, from classifying brain tumors (A, R, & S, 2019), dianosing skin cancer (M, 2019), and detecting glaucoma (Bowd & Goldbaum, 2008), to an FDA approved system alerting neurologists of acute stroke cases (Office of the Commissioner, 2019), the need to really understand what is going on under the hood can help save lives. In the field of psychiatry, we may use machine learning methods to better understand the dimensions of

subjective experience and consiousness, from using language data (very promising preliminary work is happening at Oxford University at the moment; private communications) to brain imaging studies (R. L. Carhart-Harris et al., 2012). Indeed, Stanislav Grof once remarked, "Psychedelics, used responsibly and with proper caution, would be for psychiatry what the microscope is for biology and medicine or the telescope is for astronomy" (Stanislav Grof, Hofmann, & Weil, 2008). And Johns Hopkins and Imperial College London are leading the charge in better understanding psychedelic drugs, their therapeutic effects and its relationships to dimensions of consciousness experience. With the potential of psychedelic drugs and machine learning, we may have opportunities that are previously unimaginable to alleviate suffering and better support people suffering from "abnormal" conditions such as schizophrenia and psychosis. Currently ostrasized by society, people with schizophrenia and psychosis may hold the key to better understanding the state space of the mind, and the dimensions of non-ordinary states of conscioues sness (Stanislav Grof, 2009).

With these (nascent but promising) possibilities, we must be extremely responsible with how we integrate technology, especially machine learning, into our study. Becoming intimately familiar with tools like LIME, and holding a thoughful and cautiously optimistic approach, might just be the key for us to co-create a future with less suffering.

12. Gratefulness: Acknowledgements of Foundational Connectness

In reflecting who has touched my life that led to the completing the thesis, the list becomes so long that it can fill volumes. I feel incredibly connected to the people who have shown kindness towards me, without whom I would not be here typing these words.

First and for most, I am deeply grateful for my parents, Susan Zeng and Lihong Zhao, who have been unwavering pillars for me for 23 years and counting, providing unconditional love through thick and thin, sadness and happiness. I can say for certain I would not have survived to this day without their infinite love, kindness, wisdom, understanding, and nourishment, much less complete this thesis. I am grateful for my sister, Megan Zhao, who has provided the appropriate amount of sibling rivalry, inside jokes, and support as a sister and dear friend.

I would also like to express my deep gratitude for Janis Ripoll Garriga, whose kindness, love, compassion, and presence is unparalleled. Multiple sleepless nights have been bearable only with your indefatigable support and motivation, which has kept the lights on metaphorically and literally. Thank you for being always being there for me, and for believing in me when I didn't even believe in myself. Thank you for the delightful memories, the present, and the future; I am so grateful that you are in my life, and I love you with all my heart.

Ken Kubota is one of the most special people in my life. I have Ken to thank for instilling in me deep foundational values of radical transparency, openness, and an unwavering mindful presence with people. Working with Ken for more than a year has been transformative for my psyche, outlook on life, and how I strive to be present with people. He is also the first person to make me aware of psychedelics, during one of the darkest times of my life. With his guidance, I became aware of John's Hopkin's research on psychedelics, which to put lightly, changed my life for the better. Thank you, Ken, for all that you are and all that you inspire.

I owe a deep debt of gratitude to professor Chris Callion-Burch and Lyle Ungar, who are among the best thesis advisors one can ask for. Professor Callison-Burch, thank you for being my first introduction to the exciting world of natural language processing, and for your enthusiasm, kindness, and understanding. I am so grateful for your support both inside the classroom and outside of the classroom, putting people before academics while helping me understand the kleidescope that is Natural Language Processing. Professor Lyle Ungar, thank you for hours of enlightening lectures and among the most enjoyable and inspiring conversations I have ever had at Penn. Your mind is one to behold, and the way your integrate vasts amount of information from seemingly disparate disciplines is absolutely delightful. Thank you both for your tireless support, midnight email communications, and inspiring ideas to keep me on the right track. Hopefully this thesis is passable!

Professor Linda Zhao is one of the most important mentors in my life, and is the role model who first taught me data science. Before

meeting Linda, I had virtually no idea how to program in python, and had no knowledge of `R`. You have inspired in me a deep appreciation for the beauty of statistics and the promise of data science. Thank you for believing in me so deeply and treating me like one of your own. I will never forget your presence, wisdom, love, and compassion.

Yev Barkalov deserves a very special mention: he was instrumental in the completion of this thesis. While the thesis is independent work, hours of stimulating conversation with Yev laid the groundwork for EveryQualia, which was a key inspiration for this thesis. Yev also wrote the original script for obtaining the trip reports from Erowid, which was a crucial first step to the thesis. Beyond his insane work ethic, Yev has an incredibly active mind and can always be trusted to provide a nuanced view on anything, and he is a friend I would turn to for the rest of my life for laid back passtime and intense work sessions alike.

I would also like to thank Professor Sampath Kannan for being a teacher and advisor who has lent a hand when it counts. Professor Chris Murphy, one of the most understanding and foreward thinking professors I have ever met, provided invaluable support in ensuring my thesis was completed, and serves as an inspiration for how important conversations around mental health and diversity are to students, professors, and entire institutions. Professor Brittany Shields and Professor Hilary Gerstein, thank you for your passion and nuanced take on ethics, for expanding my perspective and reminding me that any engineering and science endeavor is inevitably linked to society, and the importance contextualizing studies through ethical and societal lens. Thank you Professor Max Mintz for being a strict and thoughtful advisor — your commitment to education has instilled me a respect for propriety. Thank you David Yaden for being a sounding board for psychedelics reearch from the very beginning. Your advice and perspective were instrumental to the formation of the Penn Society for Psychedelic Science (PSPS). And Desirae Caesar, what would the computer science department do without you? You are such an instrumental part of holding the department together, with your expert knowledge and devotion to students. I know I would not be graduating without your indefatigable support; your smile and laughter will be forever committed to my memory as a source of joy.

Daniel Colson, Anders Sanberg, Robin Hanson, David Champion, Luke Sabor, Alex K. Chen, Trize Stephen Pons, Amanda Ngo, and Quintin Freriches — these are among the most brilliant minds that I have come across. Thank you for hours and hours of stimulating conversation, for keeping me inspired and pushing me to explore new horizons. Every conversation with you is delightful — it is almost always unclear how our conversations would start and where they would go, but it is beyond reasonable doubt they will be fun and challenge my thinking in unexpected ways.

Thank you to Rick Doblin and the Multidisciplinary Association for Psychedelic Studies (MAPS) for providing invaluable advice and being an unwavering source of grounded inspiration. Without you, psychedelics would not be re-entering academia with such grounded fervor as a legitimate topic of research. Thank you Erowid, especially Fire Erowid and Earth Erowid, for working tirelessly to curate such a trusted and vast source of information, all provided for free to the benefit of human kind. Thank you Qualia Research Institute for the work you are doing to formalize consciousness research. Your insights and intuitions are so incredibly inspiring, and I haven't seen anyone else do the work you are doing.

Last but certainly not least, my sincerest and deepest gratitude for the Penn Society for Psychedelic Science (PSPS), the Intercollegiate Psychedelics Network (IPN), and everyone who is involved in this growing mycelial field of connectedness. Both PSPS and IPN are such magical entities that seem to hold promise for the next wave of psychedelic renaissance. Of particular note, Victor Acero, Rahul Sood, Emily Cribas, Margaret Fleming, Kenneth Shinozuka, Aidan Fitzsimmons, Elin Ahlstrand, and Wendi Yan are simply magical. Hours of laughter and conversation in trying to figure out how to grow the mycelial network together are ones I wouldn't trade for anything else. These people, along with so many other student leaders of IPN, are so hardworking, thoughtful, inspiring; these are people I would turn to for the most difficult questions and the craziest ideas, because I trust so deeply that no matter what they will hold space thougtfully, work comfortably with uncertainty, and share their brilliant minds with radical transparency.

cc:

University of Pennsylvania (Penn), Harvard University, Princeton University, University of Connecticut (UConn), Texas Tech University (TTU), University College London (UCL), UC Boulder, Ohio State University (OSU), California Institute of Integral Studies (CIIS), Georgia State University, Colorado State University-Fort Collins, UC Colorado Springs, Yale University, Stanford University, Stanford Business School, West Virginia University, University of Greenwich, University of Oxford

The Mycelial Network
as of December 7, 2019
{ roughly in order of sporing }
Intercollegiate Psychedelics Network (IPN)

NOS OMNES MYCELIS

13. References

A, M. V., R, D. P., & S, M. S. (2019). Brain Tumor Classification Using Various Machine Learning Algorithms. >International Journal of Research in Arts and Science, 5(Special Issue), 258–270. >https://doi.org/10.9756/bp2019.1002/25

Atasoy, S., Donnelly, I., & Pearson, J. (2016). Human brain networks function in connectome-specific harmonic waves. >Nature Communications, 7(1). https://doi.org/10.1038/ncomms10340

Bellman, R., & Karush, R. (1964). Dynamic programming: a bibliography of theory and application (No. RM-3951-1-PR). >RAND CORP SANTA MONICA CA.

Blei, D. M. (2012). Probabilistic topic models. Communications of the ACM, 55(4), 77. >https://doi.org/10.1145/2133806.2133826

BOWD, C., & GOLDBAUM, M. H. (2008). Machine Learning Classifiers in Glaucoma. Optometry and Vision Science, 85(6), >396–405. https://doi.org/10.1097/opx.0b013e3181783ab6

Carhart-Harris, R. L., & Friston, K. J. (2019). REBUS and the Anarchic Brain: Toward a Unified Model of the Brain >Action of Psychedelics. Pharmacological Reviews, 71(3), 316–344. https://doi.org/10.1124/pr.118.017160

Carhart-Harris, R. L., Erritzoe, D., Williams, T., Stone, J. M., Reed, L. J., Colasanti, A., … Nutt, D. J. (2012). >Neural correlates of the psychedelic state as determined by fMRI studies with psilocybin. Proceedings of the National >Academy of Sciences, 109(6), 2138–2143. https://doi.org/10.1073/pnas.1119598109

Carhart-Harris, Robin L., Leech, R., Hellyer, P. J., Shanahan, M., Feilding, A., Tagliazucchi, E., … Nutt, D. (2014). >The entropic brain: a theory of conscious states informed by neuroimaging research with psychedelic drugs. Frontiers >in Human Neuroscience, 8, 20. https://doi.org/10.3389/fnhum.2014.00020

Carhart-Harris, Robin L, Roseman, L., Bolstridge, M., Demetriou, L., Pannekoek, J. N., Wall, M. B., … Nutt, D. J. > (2017). Psilocybin for treatment-resistant depression: fMRI-measured brain mechanisms. Scientific Reports, 7(1). >https://doi.org/10.1038/s41598-017-13282-7

Doblin, R. (2002). A Clinical Plan for MDMA (Ecstasy) in the Treatment of Posttraumatic Stress Disorder (PTSD): >Partnering with the FDA. Journal of Psychoactive Drugs, 34(2), 185–194. >https://doi.org/10.1080/02791072.2002.10399952

Edsinger, E., & Dölen, G. (2018). A Conserved Role for Serotonergic Neurotransmission in Mediating Social Behavior in >Octopus. Current Biology, 28(19), 3136-3142.e4. https://doi.org/10.1016/j.cub.2018.07.061

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust >automated machine learning. In Advances in neural information processing systems (pp. 2962-2970).

Griffiths, Roland R, Johnson, M. W., Carducci, M. A., Umbricht, A., Richards, W. A., Richards, B. D., … Klinedinst, >M. A. (2016). Psilocybin produces substantial and sustained decreases in depression and anxiety in patients with >life-threatening cancer: A randomized double-blind trial. Journal of Psychopharmacology, 30(12), 1181–1197. >https://doi.org/10.1177/0269881116675513

Griffiths, R. R., Richards, W. A., McCann, U., & Jesse, R. (2006). Psilocybin can occasion mystical-type experiences >having substantial and sustained personal meaning and spiritual significance. Psychopharmacology, 187(3), 268–283. >https://doi.org/10.1007/s00213-006-0457-5

Griffiths, T. L., Steyvers, M., Blei, D. M., & Tenenbaum, J. B. (2005). Integrating topics and syntax. In Advances in >neural information processing systems (pp. 537-544).

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. >18). New York: springer.Chicago

Johns Hopkins University. (2019, September 4). Johns Hopkins launches center for psychedelic research. Retrieved

>November 11, 2019, from The Hub website: https://hub.jhu.edu/2019/09/04/hopkins-launches-psychedelic-center/

Johnson, M. E. (2016, November 16). Principia Qualia – Opentheory.net. Retrieved December 23, 2019, from >Opentheory.net website: https://opentheory.net/2016/11/principia-qualia/

Luhn, H. P. (1957). A Statistical Approach to Mechanized Encoding and Searching of Literary Information. IBM Journal >of Research and Development, 1(4), 309–317. https://doi.org/10.1147/rd.14.0309

Manning, C. D., Raghavan, P., Hinrich Schütze, & University Of Cambridge. (2009). Introduction to information >retrieval. Cambridge: Cambridge University Press.

Mithoefer, M. C., Wagner, M. T., Mithoefer, A. T., Jerome, L., & Doblin, R. (2010). The safety and efficacy of ±3,4-methylenedioxymethamphetamine-assisted psychotherapy in subjects with chronic, >treatment-resistant posttraumatic stress disorder: the first randomized controlled pilot study. >Journal of Psychopharmacology, 25(4), 439–452. https://doi.org/10.1177/0269881110378371

Moreno, F. A., Wiegand, C. B., Taitano, E. K., & Delgado, P. L. (2006). Safety, Tolerability, and Efficacy of >Psilocybin in 9 Patients With Obsessive-Compulsive Disorder. The Journal of Clinical Psychiatry, 67(11), 1735–1740. >https://doi.org/10.4088/jcp.v67n1110

M, V. M. (2019). Melanoma Skin Cancer Detection using Image Processing and Machine Learning. International Journal of >Trend in Scientific Research and Development, Volume-3(Issue-4), 780–784. https://doi.org/10.31142/ijtsrd23936

Nichols, D. E. (2016). Psychedelics. Pharmacological Reviews, 68(2), 264–355. https://doi.org/10.1124/pr.115.011478

Office of the Commissioner. (2019). FDA permits marketing of clinical decision support software for alerting >providers of a potential stroke in patients. Retrieved December 23, 2019, from U.S. Food and Drug Administration >website: https://www.fda.gov/news-events/press-announcements/fda-permits-marketing-clinical-decision-support->software-alerting-providers-potential-stroke

Osório, F. de L., Sanches, R. F., Macedo, L. R., dos Santos, R. G., Maia-de-Oliveira, J. P., Wichert-Ana, L., … >Hallak, J. E. (2015). Antidepressant effects of a single dose of ayahuasca in patients with recurrent depression: a >preliminary report. Revista Brasileira de Psiquiatria, 37(1), 13–20. https://doi.org/10.1590/1516-4446-2014-1496

Pahnke, W. N. (1963). Drugs and mysticism: An analysis of the relationship between psychedelic drugs and the mystical >consciousness: A thesis (Doctoral dissertation, Harvard University).

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). Why should i trust you?: Explaining the predictions of any >classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining >>(pp. 1135-1144). ACM.

Rogan, J. (2018, June 29). The Joe Rogan Experience: The Human Erowid - Hamilton Morris - Podcast Notes. Retrieved >December 23, 2019, from Podcast Notes website: https://podcastnotes.org/2018/06/28/morris/

Sewell, R. A., Halpern, J. H., & Pope, H. G. (2006). Response of cluster headache to psilocybin and LSD. Neurology, >66(12), 1920–1922. https://doi.org/10.1212/01.wnl.0000219761.05466.43

SPARCK JONES, K. (1972). A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL. Journal >of Documentation, 28(1), 11–21. https://doi.org/10.1108/eb026526

Stanislav Grof. (2009). LSD : doorway to the numinous : the groundbreaking psychedelic research into realms of the >human unconscious. Rochester, Vt: Park Street Press.

Stanislav Grof, Hofmann, A., & Weil, A. (2008). LSD psychotherapy. Ben Lomond, Calif.: Multidisciplinary Association >For Psychedelic Studies.

Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F., & Mueller, A. (2015). Scikit-learn. GetMobile: >Mobile Computing and Communications, 19(1), 29–33. https://doi.org/10.1145/2786984.2786995

Wallach, H. M. (2006, June). Topic modeling: beyond bag-of-words. In Proceedings of the 23rd international conference >on Machine learning (pp. 977-984). ACM.Chicago

Wilbur, W. J., & Sirotkin, K. (1992). The automatic identification of stop words. Journal of Information Science, >18(1), 45–55. https://doi.org/10.1177/016555159201800106

Zellig Sabbettai Harris. (1954). Distributional structure. New York: Verlag Nicht Ermittelbar.

14. Appendix: archived code, explanations, and miscellaneous musings

14.1 Useful Links

- Thesis Related:
  - Thesis Master Document
  - Thesis Scratch Paper
  - Thesis Codebase
- Helpful Tutorials and Tips
  - Excellent Series from `Towards Data Science`:
    - A Practitioner's Guide to Natural Language Processing (Part I): Processing & Understanding Text
    - Feature Engineering: Continuous Numeric Data
    - Feature Engineering: Categorical Data
    - Feature Engineering: Text Data Basics
    - Feature Engineering: Text Data Advanced
  - A few userful things to know about machine learning
  - LDA: Excellent Talk by Christine Doig on Topic Models
  - Markdown Cheatsheet
  - Using R and Python together; python, R dataframe interoperability; R and python pipelining
  - Translate dplyr to pandas
  - Basic pandas tutorial
  - Recommended dependencies for pandas
  - Pandas: Working with Text Data
  - sklearn reference: handy documentation, and provides broad-strokes ontology for machine learning techniques
  - Introduction to Machine Learning with sklearn
  - Sample pipeline for text feature extraction and evaluation: using `sklearn`
  - Comparison of performances of different classifiers: from `sklearn`, super cool
  - TODO: Citing sklearn
  - Contributing to sklearn
  - Manually Creating a table of contents
  - Automaticaly Creating Table of Contents
  - Python Data Science Handbook: Free Jupyter Notebooks
  - Configuring jupyter notebook with extensions
- Publishing my thesis:
  - Jupyter Notebook Viewer
  - Hiding Code Cells for Better Viewing Experience
  - Cool Preloaders

Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

- Notes to self:
  - `~/opt/anaconda3/lib/python3.7/site-packages/jupyter_contrib_nbextensions/nbextensions/toc2` path to nbextensions toc2
  - fixing nbextensions

14.2 Archived Code

In [34]:

```
# ### Create simple wordclouds based on term frequency

# # a list of the "classical psychedelics"
# substances_psychedelics_list = ["substance_mushrooms", "substance_lsd", "substance_mescaline", "
substance_5_meo_dmt", "substance_dmt"]

# # NOTE: change this line of code to change which substances to plot
# substances_to_plot = substances_psychedelics_list
# num_plots = len(substances_to_plot)
# print("Now plotting wordclouds for {} substances...".format(num_plots))

# # creating subplots: https://stackoverflow.com/questions/25239933/how-to-add-title-to-subplots-i
n-matplotlib
# # changing figure size: https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of
-figures-drawn-with-matplotlib
# fig = plt.figure(figsize=(50, 10))
# fig.suptitle("Word Clouds based on word Frequencies", fontsize=16)

# for index, substance in enumerate(substances_to_plot):
#     # WordCloud also takes a stopwords = [] as argument; omitted.
#     wordcloud = WordCloud(max_font_size=50, max_words=100,
background_color="white").generate(substance_reports_map[substance])
#     ax = plt.subplot(num_plots, 1, index+1) # note here for subplots index must be [1,
num_plots]
#     ax.set_title("{} Wordcloud".format(substance))
#     plt.imshow(wordcloud, interpolation="bilinear")
#     plt.axis("off")

# plt.savefig('wordclouds_naive_psychedelics_tf.png')
# plt.show()


# ### Plotting a single wordcloud
# # wordcloud = WordCloud(stopwords=stopwords_all, max_font_size=50, max_words=100,
background_color="white").generate(reports_text_tokenized["substance_mushrooms"])
# # plt.figure()
# # plt.imshow(wordcloud, interpolation="bilinear")
# # plt.axis("off")
# # plt.show()
```

In [242]:

```
# # helper functions to extract keywords from documents
# # from http://kavita-ganesan.com/extracting-keywords-from-text-tfidf/#.XeLcAzJKjmE
# def sort_coo(coo_matrix):
#     tuples = zip(coo_matrix.col, coo_matrix.data)
#     return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)

# def extract_topn_from_vector(feature_names, sorted_items, topn=10):
#     """get the feature names and tf-idf score of top n items"""

#     #use only topn items from vector
#     sorted_items = sorted_items[:topn]

#     score_vals = []
#     feature_vals = []

#     # word index and corresponding tf-idf score
#     for idx, score in sorted_items:

#         #keep track of feature name and its corresponding score
#         score_vals.append(round(score, 3))
#         feature_vals.append(feature_names[idx])
```

```
#     #create a tuples of feature,score
#     #results = zip(feature_vals,score_vals)
#     results= {}
#     for idx in range(len(feature_vals)):
#         results[feature_vals[idx]]=score_vals[idx]

#     return results
```

In [244]:

```
# # Tutorial on how to extract key words based on tf-idf: http://kavita-ganesan.com/extracting-key
words-from-text-tfidf/#.XeLcAzJKjmE

# # Using the tf-idf frequencies
# feature_names_tfidf = vectorizer_wc_tfidf.get_feature_names()


# sorted_items=sort_coo(X_vectors_tfidf.tocoo())

# #extract only the top n; n here is 10
# keywords=extract_topn_from_vector(feature_names_tfidf,sorted_items,10)

# # now print the results
# print("\n=====Doc=====")
# # print(doc[:1000])
# print("\n===Keywords===")
# for k in keywords:
#     print(k,keywords[k])


# # # get the first vector out (for the first document)
# # first_vector_tfidfvectorizer= X_vectors_tfidf[0]

# # # place tf-idf values in a pandas data frame
# # df = pd.DataFrame(first_vector_tfidfvectorizer.T.todense(), index=feature_names_tfidf, columns
=["tfidf"])
# # df.sort_values(by=["tfidf"],ascending=False)
```

```
=====Doc=====

===Keywords===
kava 0.726
kratom 0.568
like 0.335
ibogaine 0.371
dmt 0.368
seeds 0.362
salvia 0.359
```

PCA CODE

Other ways of Visualizing Trip reports:

- Clustering with ordinary `K-means` and `minibatch K-means`; see sklearn example
- `Principle Components Analysis (PCA)` of trip reports

In [0]:

```
# TODO: Example code from: https://scikit-
learn.org/stable/auto_examples/text/plot_document_clustering.html#sphx-glr-auto-examples-text-plot
-document-clustering-py

# Author: Peter Prettenhofer <peter.prettenhofer@gmail.com>
#         Lars Buitinck
# License: BSD 3 clause
from sklearn.datasets import fetch_20newsgroups
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn import metrics

from sklearn.cluster import KMeans, MiniBatchKMeans

import logging
from optparse import OptionParser
import sys
from time import time

import numpy as np


# Display progress logs on stdout
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s %(levelname)s %(message)s')

# parse commandline arguments
op = OptionParser()
op.add_option("--lsa",
              dest="n_components", type="int",
              help="Preprocess documents with latent semantic analysis.")
op.add_option("--no-minibatch",
              action="store_false", dest="minibatch", default=True,
              help="Use ordinary k-means algorithm (in batch mode).")
op.add_option("--no-idf",
              action="store_false", dest="use_idf", default=True,
              help="Disable Inverse Document Frequency feature weighting.")
op.add_option("--use-hashing",
              action="store_true", default=False,
              help="Use a hashing feature vectorizer")
op.add_option("--n-features", type=int, default=10000,
              help="Maximum number of features (dimensions)"
                   " to extract from text.")
op.add_option("--verbose",
              action="store_true", dest="verbose", default=False,
              help="Print progress reports inside k-means algorithm.")

print(__doc__)
op.print_help()


def is_interactive():
    return not hasattr(sys.modules['__main__'], '__file__')


# work-around for Jupyter notebook and IPython console
argv = [] if is_interactive() else sys.argv[1:]
(opts, args) = op.parse_args(argv)
if len(args) > 0:
    op.error("this script takes no arguments.")
    sys.exit(1)


# #############################################################################
# Load some categories from the training set
categories = [
    'alt.atheism',
    'talk.religion.misc',
    'comp.graphics',
    'sci.space',
]
# Uncomment the following to do the analysis on all the categories
# categories = None

print("Loading 20 newsgroups dataset for categories:")
print(categories)

dataset = fetch_20newsgroups(subset='all', categories=categories,
                             shuffle=True, random_state=42)

print("%d documents" % len(dataset.data))
print("%d categories" % len(dataset.target_names))
print()

labels = dataset.target
```

```
true_k = np.unique(labels).shape[0]

print("Extracting features from the training dataset "
      "using a sparse vectorizer")
t0 = time()
if opts.use_hashing:
    if opts.use_idf:
        # Perform an IDF normalization on the output of HashingVectorizer
        hasher = HashingVectorizer(n_features=opts.n_features,
                                   stop_words='english', alternate_sign=False,
                                   norm=None, binary=False)
        vectorizer = make_pipeline(hasher, TfidfTransformer())
    else:
        vectorizer = HashingVectorizer(n_features=opts.n_features,
                                       stop_words='english',
                                       alternate_sign=False, norm='l2',
                                       binary=False)
else:
    vectorizer = TfidfVectorizer(max_df=0.5, max_features=opts.n_features,
                                 min_df=2, stop_words='english',
                                 use_idf=opts.use_idf)
X = vectorizer.fit_transform(dataset.data)

print("done in %fs" % (time() - t0))
print("n_samples: %d, n_features: %d" % X.shape)
print()

if opts.n_components:
    print("Performing dimensionality reduction using LSA")
    t0 = time()
    # Vectorizer results are normalized, which makes KMeans behave as
    # spherical k-means for better results. Since LSA/SVD results are
    # not normalized, we have to redo the normalization.
    svd = TruncatedSVD(opts.n_components)
    normalizer = Normalizer(copy=False)
    lsa = make_pipeline(svd, normalizer)

    X = lsa.fit_transform(X)

    print("done in %fs" % (time() - t0))

    explained_variance = svd.explained_variance_ratio_.sum()
    print("Explained variance of the SVD step: {}%".format(
        int(explained_variance * 100)))

    print()


# #############################################################################
# Do the actual clustering

if opts.minibatch:
    km = MiniBatchKMeans(n_clusters=true_k, init='k-means++', n_init=1,
                         init_size=1000, batch_size=1000, verbose=opts.verbose)
else:
    km = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1,
                verbose=opts.verbose)

print("Clustering sparse data with %s" % km)
t0 = time()
km.fit(X)
print("done in %0.3fs" % (time() - t0))
print()

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels, km.labels_))
print("Completeness: %0.3f" % metrics.completeness_score(labels, km.labels_))
print("V-measure: %0.3f" % metrics.v_measure_score(labels, km.labels_))
print("Adjusted Rand-Index: %.3f"
      % metrics.adjusted_rand_score(labels, km.labels_))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, km.labels_, sample_size=1000))

print()


if not opts.use_hashing:
    print("Top terms per cluster:")
```

```
    if opts.n_components:
        original_space_centroids = svd.inverse_transform(km.cluster_centers_)
        order_centroids = original_space_centroids.argsort()[:, ::-1]
    else:
        order_centroids = km.cluster_centers_.argsort()[:, ::-1]

    terms = vectorizer.get_feature_names()
    for i in range(true_k):
        print("Cluster %d:" % i, end='')
        for ind in order_centroids[i, :10]:
            print(' %s' % terms[ind], end='')
        print()
```

```
Downloading 20news dataset. This may take a few minutes.
2019-11-30 00:37:12,265 INFO Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
2019-11-30 00:37:12,270 INFO Downloading dataset from
https://ndownloader.figshare.com/files/5975967 (14 MB)
```

```
Automatically created module for IPython interactive environment
Usage: ipykernel_launcher.py [options]

Options:
  -h, --help            show this help message and exit
  --lsa=N_COMPONENTS    Preprocess documents with latent semantic analysis.
  --no-minibatch        Use ordinary k-means algorithm (in batch mode).
  --no-idf              Disable Inverse Document Frequency feature weighting.
  --use-hashing         Use a hashing feature vectorizer
  --n-features=N_FEATURES
                        Maximum number of features (dimensions) to extract
                        from text.
  --verbose             Print progress reports inside k-means algorithm.
Loading 20 newsgroups dataset for categories:
['alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space']
3387 documents
4 categories

Extracting features from the training dataset using a sparse vectorizer
done in 1.086469s
n_samples: 3387, n_features: 10000

Clustering sparse data with MiniBatchKMeans(batch_size=1000, compute_labels=True, init='k-
means++',
                init_size=1000, max_iter=100, max_no_improvement=10,
                n_clusters=4, n_init=1, random_state=None,
                reassignment_ratio=0.01, tol=0.0, verbose=False)
done in 0.154s

Homogeneity: 0.539
Completeness: 0.572
V-measure: 0.555
Adjusted Rand-Index: 0.576
Silhouette Coefficient: 0.007

Top terms per cluster:
Cluster 0: god people jesus say don believe christian bible com religion
Cluster 1: graphics university image thanks ac files uk file com 3d
Cluster 2: space com nasa access henry digex gov article pat toronto
Cluster 3: sandvik keith sgi com livesey kent caltech apple newton solntze
```

Multilabel learning code templates

In [0]:

```
## TODO: Example code from sklearn, see: https://scikit-
learn.org/stable/auto_examples/plot_multilabel.html#multilabel-classification
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_multilabel_classification
from sklearn.multiclass import OneVsRestClassifier
```

```
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import CCA


def plot_hyperplane(clf, min_x, max_x, linestyle, label):
    # get the separating hyperplane
    w = clf.coef_[0]
    a = -w[0] / w[1]
    xx = np.linspace(min_x - 5, max_x + 5)  # make sure the line is long enough
    yy = a * xx - (clf.intercept_[0]) / w[1]
    plt.plot(xx, yy, linestyle, label=label)


def plot_subfigure(X, Y, subplot, title, transform):
    if transform == "pca":
        X = PCA(n_components=2).fit_transform(X)
    elif transform == "cca":
        X = CCA(n_components=2).fit(X, Y).transform(X)
    else:
        raise ValueError

    min_x = np.min(X[:, 0])
    max_x = np.max(X[:, 0])

    min_y = np.min(X[:, 1])
    max_y = np.max(X[:, 1])

    classif = OneVsRestClassifier(SVC(kernel='linear'))
    classif.fit(X, Y)

    plt.subplot(2, 2, subplot)
    plt.title(title)

    zero_class = np.where(Y[:, 0])
    one_class = np.where(Y[:, 1])
    plt.scatter(X[:, 0], X[:, 1], s=40, c='gray', edgecolors=(0, 0, 0))
    plt.scatter(X[zero_class, 0], X[zero_class, 1], s=160, edgecolors='b',
                facecolors='none', linewidths=2, label='Class 1')
    plt.scatter(X[one_class, 0], X[one_class, 1], s=80, edgecolors='orange',
                facecolors='none', linewidths=2, label='Class 2')

    plot_hyperplane(classif.estimators_[0], min_x, max_x, 'k--',
                    'Boundary\nfor class 1')
    plot_hyperplane(classif.estimators_[1], min_x, max_x, 'k-.',
                    'Boundary\nfor class 2')
    plt.xticks(())
    plt.yticks(())

    plt.xlim(min_x - .5 * max_x, max_x + .5 * max_x)
    plt.ylim(min_y - .5 * max_y, max_y + .5 * max_y)
    if subplot == 2:
        plt.xlabel('First principal component')
        plt.ylabel('Second principal component')
        plt.legend(loc="upper left")


plt.figure(figsize=(8, 6))

X, Y = make_multilabel_classification(n_classes=2, n_labels=1,
                                      allow_unlabeled=True,
                                      random_state=1)

plot_subfigure(X, Y, 1, "With unlabeled samples + CCA", "cca")
plot_subfigure(X, Y, 2, "With unlabeled samples + PCA", "pca")

X, Y = make_multilabel_classification(n_classes=2, n_labels=1,
                                      allow_unlabeled=False,
                                      random_state=1)

plot_subfigure(X, Y, 3, "Without unlabeled samples + CCA", "cca")
plot_subfigure(X, Y, 4, "Without unlabeled samples + PCA", "pca")

plt.subplots_adjust(.04, .02, .97, .94, .09, .2)
plt.show()
```

In [ ]:

```
`TODO`

- Baseline: multinomial Bayes and other techniques; see sklearn [classifying text documents](https://scikit-learn.org/stable/auto_examples/text/plot_document_classification_20newsgroups.html#sphx-glr-auto-examples-text-plot-document-classification-20newsgroups-py)
- Baseline: Logistic Regression:
    - Coefficients returned by logistic regression can be used as weights to scale words in worclouds
    - TODO: How do we handle the multilabel case?
    - Appropriate regularization like LASSO / Ridge (Elastic Net) for feature selection
- Baseline: Naive Bayes classifier
- BERT fine-tuning
    - [Huggingface](https://github.com/huggingface/transformers) + transformers - See Chris Callison-Burch's recommendations
- Handling `multilabel learning`, see
```

Other Miscellaneous Code and Musing Fragments

In [37]:

```
# # Explore the dataset trips.csv
# import csv
# printed = 0
# with open('trips.csv') as f:
#   csv_reader = csv.reader(f)
#   for line in csv_reader:
#     if printed < 2:
#         print(line)
#         printed += 1
#     else:
#         break
```

14.3 Issues Encoutered, and solutions

- Python not working:
  `brew install python`
  `brew update python`
- Need to update xcode command line tools after every major / minor os upgrade: `xcode-select --install` ; see here
- Issues with installing wordcloud
- TODO: Write an article explaining managing python versions on Mac, using both `Anacoda` and `pip`
- Cannot do `brew link <packageName>` ; see here
- Adding `autocompletion` to jupyter notebook with this tool
- Could not for the life of me get `toc2` to work
- 5 tries... with trying to get code hiding to work; 2019-12-04, still no luck...
- `TODO` : Putting a pin in this for now rip
- see this

14.4 Fun (Optional) things discovered while working on Project

- Add animated progress nav
- New York Times recommendation system
- CSS background tricks
- spaCy webapplication demo
- spaCy NER visulizer demo
- NLP with spaCy course

14.5 Originally Planned Conclusion

The following sections were originally planned for the thesis. A decision was made to omit them in this submission, with plans of publishing them at a later date, to be made available on `alextzhao.io` and `paradiseinstitute.org`.

12. Meta-discussions: Neuroscience, Ethics, Mycelial Connectedness, and Future Directions - The Truly Important Stuff

12.1 What is Language

12.2 What is (Big) Data

12.3 What is Data Privacy and Security

12.4 What is Reproducibility

12.5 Open Science: A Proposal for Collaborative Research

12.6 Towards Mycelial Connectedness

12.7 The Institute of Paradise Engineering

In [0]:

```
# Settings for nbviewer; can safely ignore
# %%html
# <script
src="https://cdn.rawgit.com/parente/4c3e6936d0d7a46fd071/raw/65b816fb9bdd3c28b4ddf3af602bfd6015486
ode_toggle.js"></script>
```