UNIVERSITÀ DEGLI STUDI DI MILANO

MASTERS'S PROGRAM IN COMPUTER SCIENCE

Word embeddings on Songs Metadata for Genre Recommendation

by

Shripad Ambure

Thesis submitted for the degree of master's in *Computer Science* (39° credits)

February 2023

Prof. Paolo Ceravalo — Supervisor

Dr. Samira Maghool — Co-Supervisor

UNIVERSITÀ
DEGLI STUDI
DI MILANO

DIPARTIMENTO DI INFORMATICA
(FACOLTA' DI SCIENZE E TECNOLOGIE)

Keywords: Word Embeddings, Doc2Vec, Spotify Songs, Genre-subgenre, Metadata, Recommender Systems, Languages, Lyrics, PCA, NLP, Recurrent Neural Network(RNN), Cluster analysis, Kmeans clustering.

To My Parents . . .

Declaration

I hereby certify that the work presented in the thesis titled **"Word embeddings on Songs metadata for genre recommendation"**, which was submitted to the University of Milan, in partial fulfillment of the requirement for the award of the degree of Master's in Computer Science (Informatica), is an original research project conducted under the supervision of **Prof. Paolo Ceravalo** and **Dr. Samira Maghool**.

This Thesis represents my own work which has been done after registration for the degree of Master's at the University of Milan and has not been previously included in a thesis or dissertation submitted to this or any other institution for a degree, diploma, or other qualifications.

Shripad Ambure

February 2023

Abstract

We Present an ML Approach to the music recommendation system, using the Doc2Vec embedding techniques and an unsupervised learning algorithm. Machines must be able to perceive sounds in the same way that humans do. They must be familiar with diverse sounds, be able to associate them with actual items, entities, or events, recognize and categorize them, and know or be able to find links between them. Acoustic Intelligence is what we term this intelligence.

Currently, billions of music songs are accessible on the online platform. Therefore, a music recommendation system might be beneficial for selecting and categorizing music tracks according to consumers' demands. To construct a recommendation system, we require a large quantity of data together with user preference information.

Due to the quick expansion of music tracks in today's world, both offline and online, better access to automatically identify a song's genre, and based on that, the user will have a fantastic experience. As a result, this study has proposed a method that uses deep learning to do genre classification on the music using word embedding to select songs based on the metadata information. Having an extensive collection and categorizing the songs according to their genre using the doc2vec model for constructing vectors and then finding songs with similar contexts and providing suggestions. Consequently, the proposed system functions as

a complete music recommendation system.

This Thesis's disciplinary environment includes music Recommendations over the Spotify Genres, Languages, and Lyrics, namely, from which data may be collected for audio signal analysis. Our primary objective is to research Spotify music data and its genres and languages. Later, we use unsupervised learning algorithms and clustering techniques to construct song vectors' averages and then distances for each of the music genres. Clearly, the less distant clusters are those songs that could be suggested by a recommender system.

Acknowledgements

First and foremost I'd like to thank **myself** and I'd want to express my gratitude to everyone who helped me as I worked on this master's thesis, since their contributions were vital in helping me achieve the goals set for myself. I'm grateful to my advisers, Dr. Paolo Ceravalo and Dr.Samira Mghool for their guidance. encouraging, mentoring, and assisting me in the preparation of this Master Thesis.

I also like to thank my dear friends, great buddies, for motivating me and sharing their experiences with me.

I express my sincere gratitude to my dear Shreya for her patience with me, smart advice, encouragement, and motivation. At the same time, I worked on this master's thesis and helped brighten my days through these difficult economic times. I want to show my gratitude to the whole academic family throughout my master's career.

Last but not least, I want to thank my family. I want to start by expressing my gratitude to my parents, Raju and Renuka, for their unwavering love and support throughout my life. I want to express my gratitude to my brother Karan, Sagar, and the rest of my family for their unfailing love, support, inspiration, and encouragement. They have made me smile more than anybody else in the world and cheered me up whenever I needed it.

Table of contents

List of figures

Part I

Section One

Chapter 1

Introduction

1.1 What is Music Recommender system

Increased by the rise of online music stores and music streaming services, digital music distribution has led to the ubiquitous availability of music. Music listeners, suddenly presented with an unprecedented amount of freely accessible material, might quickly become overwhelmed. Music recommender systems, the basis of this chapter, give advice to users browsing enormous collections. Music items that may be suggested include artists, albums, songs, genres, and radio stations. In this chapter, we explain the unique aspects of the music recommendation issue, as contrasted to other content domains, such as books or movies. To understand the disparities, let us first evaluate the length of time necessary for a consumer to consume a single media piece. There is a big gap in consumption time between novels (days or weeks), movies (one to a few hours), and songs (usually a few minutes) (typically a few minutes). Consequently, the time it takes for a user to form views regarding music might be substantially less than in other areas, contributing to the transitory, even disposable, quality of music.

Similarly, with music, a single item may be eaten frequently (even many times in succession), but other media products are normally ingested at most a few times. This means that a user could not only accept but rather welcome suggestions of previously recognized things. On a practical level, another differentiating trait is that music may be directly handled at multiple levels of abstraction. For instance, whereas movie recommenders normally advise individual objects to the viewer, music recommendation approaches may suggest groups of songs by genre, artist, or album. From a practitioner's standpoint, we remark that collaborative filtering approaches are essentially domain-agnostic and can be simply applied to music rating data. However, in the music domain, explicit rating data is relatively uncommon, and even when accessible, tends to be sparser than in other domains. Instead, implicit positive feedback is generally taken from uninterrupted (or unrejected) listening experiences.

Due to the sparsity of easily accessible user feedback data, music recommendation algorithms tend to depend more upon content descriptions of items than strategies in other domains. Content-based music recommendation techniques are strongly tied to the broader field of music information retrieval (MIR), which aims at extracting semantic information from or about music at different representation levels (e.g., the audio signal, artist or song name, album cover, or score sheet) (e.g., the audio signal, artist or song name, album cover, or score sheet). Many of these technologies employ signal processing and analysis methods directly to music in order to extract musically relevant elements and in turn offer novel search and browsing interfaces. In all these instances, as is the case with memory based in (section 1.1) collaborative filtering approaches the idea of similarity is fundamental. For content-based techniques, item similarity is often computed between item feature vectors. In (Section 1.2) presents an introduction to content-based music recommendation strategies, covering both metadata and signal analysis methods. From the user's viewpoint, material may play a crucial role in determining preferences for music. Studies in music psychology

reveal that a user's short-term music preferences are impacted by different elements, such as the environment, the emotional state, or the activities of the user. We elaborate on contextual music recommendation systems. In (section 1.3) we describe hybrid recommendation systems which mix collaborative filtering, content-based filtering, and context-based algorithms.

Because with the launch of music applications such as Spotify and Apple Music, the world of music is spinning faster than ever. Individuals discover it quite useful for a fast quick fix as we many lack the desire or time to explore new music. way to create music recommendations for them While it is feasible to utilize an application to find music that is similar to what you enjoy, it is sometimes more convenient to do it manually. It is significantly more convenient to simply select the phone, say the name of a song, and get a selection of related songs with listening options. This is a really user-friendly function that hasn't been thoroughly explored, and any smartphone or laptop would benefit immensely from having it.

There are two methods to approach offering music suggestions. Either a collaborative filtering system or a content-based filtering method is used. When there isn't a lot of history or previous data to base new suggestions on, a content-based filtering method will be quicker computationally and simpler to use. Utilizing a rating matrix, collaborative filtering systems provide recommendations to users based on what other users with comparable interests are finding appealing. A content-based filtering system is the best option for this lyric-based recommendation system since the user is providing a piece of brand-new music to draw suggestions from and hasn't provided any prior listening history.

This report aims to suggest songs that, depending on the user's preferences, they would like and have lyrics that are comparable. The way the whole system operates is as follows. The title and artist of a song are first spoken out in their raw audio form. The song and artist

names are broadcast to the lyric-finding model by the speech model once it has picked up the audio. This model provides the name of the song and utilizes a music dataset provided by an API to extract all of the lyrics. Using a Doc2Vec embedding model, the vector representation of the song lyrics is then computed and contrasted with vector representations of all other song lyrics. Finally, cosine similarity is used to determine the similarities between the songs, and the outcome is a list of songs that are comparable based on the lyrics.

In recent years, a number of Western music streaming services have appeared, including Google Play Music, Apple Music, Last.fm, Pandora, Spotify, and others; some of these services are not yet accessible in India. These music streaming services keep track of user preferences and suggest songs to subscribers. Several music streaming services, including Apple Music, Gaana, Hungama, Saavn, and Wynk Music, have recently launched in India. Some of them only serve as a music library and do not provide song recommendations. One of the streaming video platforms that uses collaborative filtering to provide recommendations is YouTube (Davidson et al., 2010). Additionally, it offers the option to search by song title, as well as to explore a video using any buzzwords found inside a lyric body, but only if the whole lyric is included in the description.

1.2 Recommendation System Types and Use Cases

The worldwide music industry is projected to be worth $ 130 billion. The sector is predicted to expand rapidly as the popularity of music streaming services such as Spotify, YouTube Music, and Amazon Music develops. Because of the rising demand for online music streaming services, it is necessary to tailor the customer experience and give them with what they desire. A music recommendation system is a strong tool used by many streaming music service providers to assist their clients find the perfect songs and have a good music streaming experience.

1.2.1 Why is the music streaming industry dependent on recommendations?

Providers of music streaming services can provide subscribers with appropriate song suggestions in real time using music recommender systems. Offering personalization, it increases user engagement. Service providers and consumers may both benefit from the recommender system. It helps users identify and choose the ideal music more quickly, and it also aids service providers in keeping users on their platform for longer periods of time. Instead of getting lost in the many music selections on your favorite streaming app, you instantly start playing pre-made playlists with songs selected to suit your individual interests. All of this is made possible by AI-powered music recommendation systems (MRS).

This work will examine music recommendation algorithms, particularly those used by streaming music audio services. We'll study its fundamental working principles from both the technical and user perspectives, examine its advantages and commercial use cases, and back them up with concrete examples. The cherry on top will be our own collaborative recommendation system, which we'll build using a free public dataset and the K-means

clustering algorithm to choose the music you'll like. Startups in the music and audio sectors will find this work to be helpful, particularly those whose business models are based on streaming technology. Continue reading if you want to satiate your curiosity or if you are just an enthusiast interested in learning how a music recommender works and maybe even ready to put it to the test.

The purpose of a recommendation system, a sorting system, is to forecast the preference that a user will have for a certain element, in this example, a song. It is the central component of large engines that employ certain recommender algorithms and provide consumers with a single item or a group of things based on their predictions. Whether we realize it or not, a number of recommendation systems have lately ingrained themselves into our everyday lives. Starting with precisely targeted advertising product recommendations and ending with a customized movie or music playlists made just for us, recommendation systems seem to be invading our daily lives from almost every nook and cranny of the internet.

Ultimately, TikTok, a current sensation, is centered on a recommendation system engine; as a result, its algorithms are unique and promise to provide artists with many more possibilities to expand organically, or in other words, with the aid of recommender system algorithms. The primary engine of streaming applications like Spotify, YouTube Music, Spotify, Tidal, and others in the music business includes recommendation algorithms. They guarantee that you will have a high-quality streaming experience.

1.2.2 What is the process behind a music recommendation engine?

The following are the three most prevalent recommendation systems:

- **Content based Filtering** : Recommendations based on similarities between the lyrics or, in this case, other qualities of two songs
- **Collaborative Filtering** : Using matrices with ratings for each piece of content, in this case, a song, recommendations based on users' similar likes.
- **Hybrid recommendations** : To provide clients a wider selection of items, hybrid recommendation systems promote things using both content-based and collaborative filtering concurrently. This emerging recommendation system is said to provide suggestions that are more precise than those made by existing recommender systems.

The **Content-based** approach is determined by how comparable various items are. A client who subscribes to a streaming music service may create playlists by ranking tunes they like or dislike. The extraction of keywords from a user's favorite song's description, comparison of those keywords with keywords from other songs, and subsequent recommendations of songs that are similar to the user is a crucial component of a content-based recommendation system.After then, the overlapping user preferences and music ratings are combined to build a **collaborative system**. In other words, if user A appreciates a song, user B is likely to enjoy it as well, and vice versa. This is predicated on the notion that if users A and B have similar interests, they may be recommended similar music. Collaborative recommendation systems are regarded to be more accurate since they rely on direct user interactions rather than content similarities. **Detailed information about these recommendation systems will be provided in the next section.**

1.3 Content Based Filtering

Content information consists of both metadata given by outside sources and any information characterizing music components that may be derived from the audio stream (e.g., web documents, discography data, or tags). In this part, we review the literature on content-based methods for music recommendation and classify the current methods according on the sources of information they use.

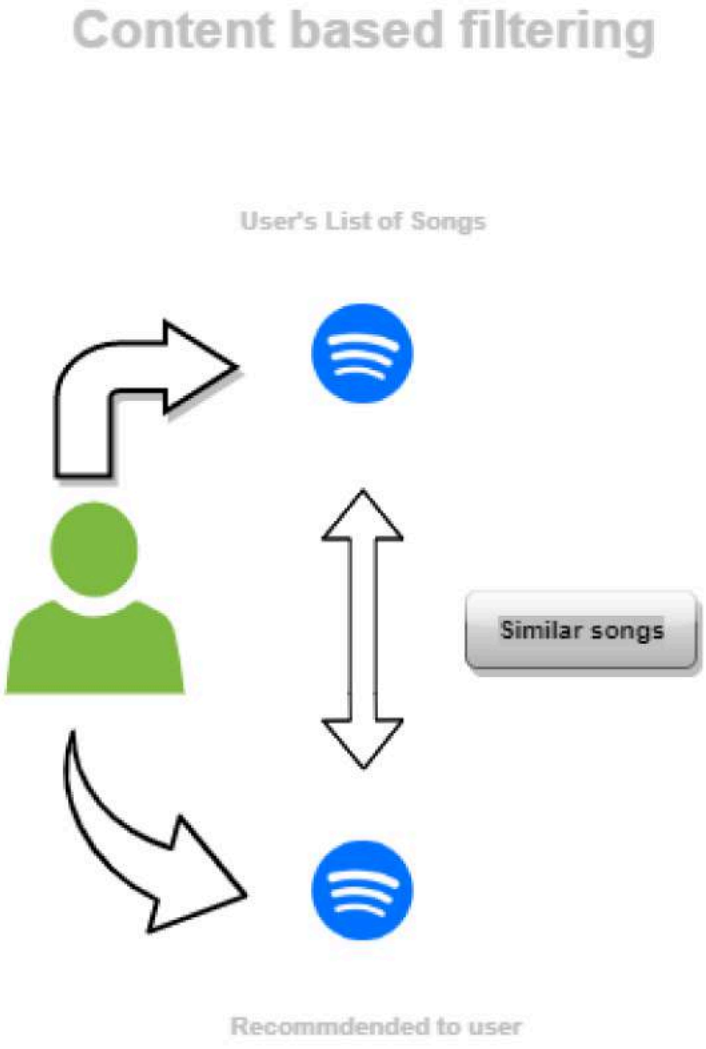


Figure 1.1 Workflow of Content-Based filtering

The study of the content of the items considered for recommendations makes up the content-based recommendation. This strategy seeks to deduce the Using a user's choices, recommendations are made for material that is similar to what they have previously enjoyed. This approach relies solely on sound similarity that is inferred from the information retrieved from previously played songs and does not require any listener input. The foundation of this approach is on the similarities between many things. It is a question of extracting characteristics that best characterize the music in order to assess similarities. The closest product to those the consumer already likes is then suggested by the machine learning algorithms.

Consequently, it is essential to develop item profiles based on characteristics derived from things. Furthermore, this approach needs user profiles that reflect their preferences and previous platform activity. These profiles will take the shape of a list of weights for each trait we've chosen, each of which indicates how important it is. The fundamental benefit of this strategy is that obscure music has an equal chance of being suggested as well-known music, whether it is timeless or popular right now. As a result, emerging artists with a few "views" can also be promoted. Additionally, by doing this, the issue of the cold start and particularly of the new things is avoided: Instead of requiring integration time as is the case with recommendation systems based on a collaborative filtering technique, new things added into the system can be suggested right away. The drawback of this approach is that it over-specializes and restricts the variety of recommendations. Additionally, it takes time for a new user to be fully integrated; they must first listen to and rate a particular number of songs before being given suggestions; this is known as a "cold start."

1.4 Collaborative Filtering

This type of recommendation is based on an analysis of the behavior of both the listeners and all other platform users. Here, it is presumpted that examining previous users' opinions might help predict another user's preferences for a certain product.

Recommendations are made to users based on other users who share their tastes but have not yet given ratings. In fact, we have been asking our friends, family, and coworkers for ideas for music, restaurants, movies, and other things throughout the years. Here, an effort is made to imitate that strategy. This method (based on user ratings) was developed by Netflix but is now widely used, especially for Spotify's Discover Weekly.

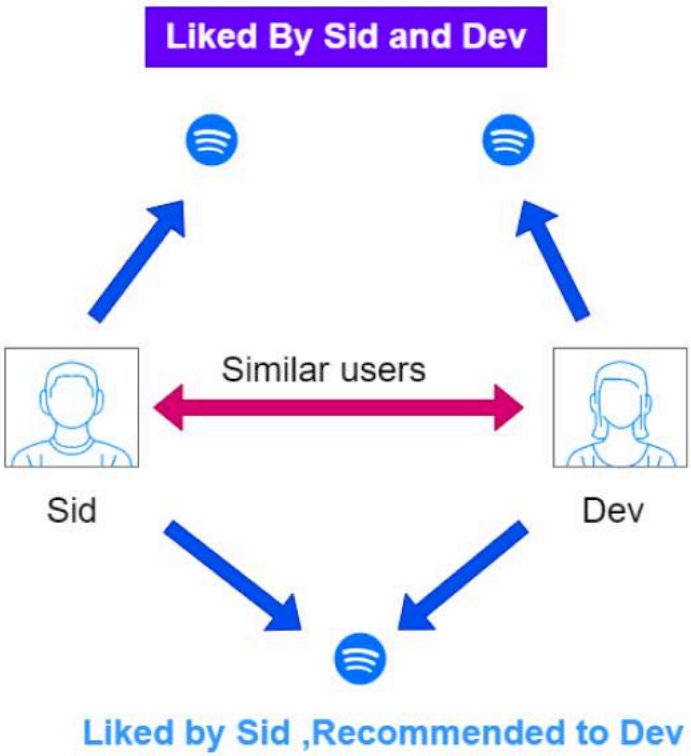


Figure 1.2 Workflow of Collaborative filtering

Memory-based approaches are the first family of collaborative filtering techniques. The idea is to keep every piece of information in a Users/Songs matrix. Feedback, whether

implicit or explicit, can help with this. If the item has been listened to at least once in the first case, the value is 1, otherwise, it is 0. If stars are present, the value in the latter is that number; otherwise, it is 0. The result is a large matrix. Spotify attempts to decrease this matrix by the inner product of two further smaller matrices.

Collaborative filtering and Spotify's recommender system are now often used interchangeably. The DSP behemoth was the first to implement the so-called "Netflix technique" in the area of music recommendations, and collaborative filtering was widely highlighted as the key component of its recommendation engine. "By looking at what other people with similar likes are listening to, we can learn which music to recommend to a user." The algorithm only analyzes users' listening histories; for example, if user A has appreciated songs X, Y, and Z, and user B has only heard X and Y, we should recommend song Z to them. Spotify can determine if two songs are similar (if similar people listen to them) and whether two users are similar by keeping a vast user-item interaction matrix spanning all users and tracks on the platform (if they listen to the same songs).

1.5 History of Spotify

Midway through the first decade of the year 2000, web designer and entrepreneur Daniel Ek developed Spotify, a cross-platform program with Swedish origins. It started off as a PC program in 2008 and then released a mobile and dynamic version for smartphones a year later. Spotify began its conquest of the music industry by extending its business and introducing services like access to specialized magazines and other related applications after signing contracts with numerous record labels, including Sony BMG, Hollywood Records, Warner Music, and Universal Music, among others. the musical industry.

As one of the most significant apps in the music business, Spotify has succeeded in connecting with its customers by upholding the freemium business model (free and premium)

and enabling use through partnerships with significant social networks (Facebook, Twitter, Instagram, Snapchat, etc.)

1.5.1 Spotify (A Case Study)

The most important on-demand music service app available today is Spotify. The company has a history of breaking technological barriers by utilizing AI and machine learning to improve the user experience through insightful analysis of complex consumer data. The introduction of the Discover Weekly Playlist, which reached 40 million people in its first year, was one significant achievement they had attained very quickly. Users receive a personalized list of thirty songs every Monday that includes songs they may not be familiar with, but the suggestions are made based on the user's search history patterns and possible musical preferences. Combining three machine learning (ML) models—Collaborative Filtering, Natural Language Processing, and Audio model allows for the creation of this playlist.

According to the algorithm for collaborative filtering, users who listen to the same songs or use similar devices are identified. It then suggests music that only one of the users has heard. Spotify has the same cold-start issue as every new product has, where there is no user data to operate on. Spotify employs CNNs (Convolutional Neural Networks) to analyze songs with similar acoustic patterns and provide recommendations by running them over the audio of a song.

It makes use of a Word2Vec method, which encodes words into vectors, in terms of NLP. Therefore, it is more likely that two vectors with similar shapes represent the same thing. The playlists are treated as a paragraph or large block of text, with each song being treated as a separate word. As a consequence, vector representations of songs are created, which may be used to compare two different musical compositions. As a result, Spotify can determine which songs are similar to one another, allowing it to address the cold start issue and propose music with few listens.

However, in our thesis project, we are utilizing the doc2vec approach, which vectorizes genre, linguistic, and lyric documents before performing NLP tasks to analyze the vectors for music recommendation.

Basically, Songs are categorized in accordance with data from raw audio recordings that is analyzed using audio models. Regardless of web publicity, this enables the platform to assess all songs and generate suggestions. For instance, if there is a new song launched on the platform by a new artist and there is little internet and social media publicity of it, NLP models can miss it. However, the collaborative filtering model will be able to analyze the track and propose it to comparable consumers alongside other more well-known songs by using song data from audio models.

Chapter 2

Deep Learning

This Chapter explains clustering techniques in Deep Learning (DL) and introduces the concept of deep learning. The type of deep learning used in this thesis is unsupervised learning, which is a major focus. To perform the task of clustering techniques on songs dataset containing the information about songs, deep learning framework uses the Doc2Vec model, which represents every document as a vector and can be generalized to represent every document as a vector. It is our main objective to analyze songs on the basis of their languages, genres, and lyrics.

2.1 What is Deep Learning

Artificial intelligence (AI) and machine learning techniques called deep learning model how people acquire specific types of information. Data science, which also encompasses statistics and predictive modeling, contains deep learning as a key component. Deep learning makes this process quicker and simpler, which is very advantageous to data scientists who are entrusted with gathering, analyzing, and interpreting massive volumes of data. Deep learning may be viewed as a means to perform predictive analytics at its most basic level. Deep learning algorithms are piled in a hierarchy of increasing the scale and abstraction,

as opposed to conventional machine learning algorithms, which are linear. A deep neural

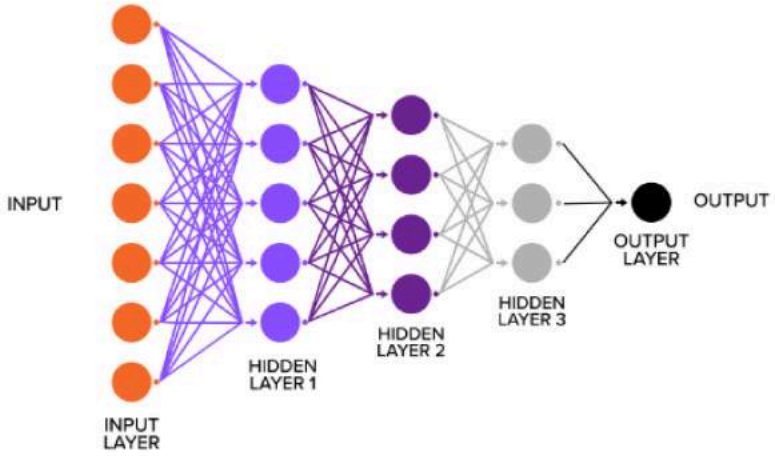


Figure 2.1 Deep Learning with hidden layers

network used for deep learning has several hidden layers with numerous nodes in each hidden layer. Deep learning creates methods for deep learning that may be used to train on complicated data and forecast results.

When feature engineering is done in advance, traditional machine learning may readily produce a forecast for data structure.

However, feature engineering with a strong model is particularly challenging and time-consuming because to the rise of unstructured data (text, photos, audio, and speech) in today's digital world. This issue is resolved via a deep learning network.

Deep learning systems may manually design features as well as learn on their own. It continually improves in effectiveness because of this mechanism.

The foundation of deep learning is the automated and gradual extraction of higher-level characteristics from the input data, which may be anything; in my case, it will be music data. The introduction of artificial intelligence to the world of music streaming has completely altered the consumer experience. Companies that offer music streaming services, like Spotify, youtube music, amazon music, and Gaana have been employing AI to analyze the

listening habits of their users and suggestions for specifically tailored playlists for a more individualized user experience. The music streaming apps use AI-based recommendation algorithms to look at listeners' prior listening behavior and suggest new tracks. One music juggernaut that has excelled in this regard is Spotify.

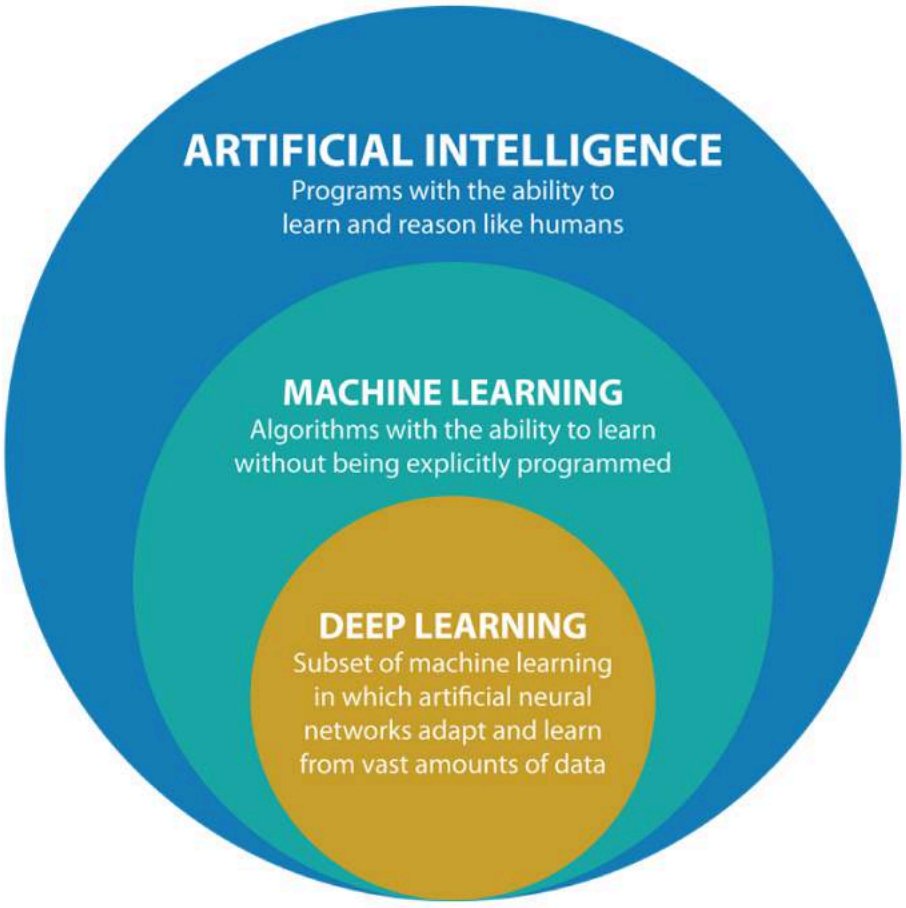


Figure 2.2 Simple Hierarchy of Deep Learning

The actual impact of artificial intelligence can be seen in the use of filtering engines, which search through tens of thousands of recently uploaded music to create playlists and personalized recommendations for each user, removing the need for listeners to sift through tens of thousands of songs to find their favorites. Additionally, AI filtering engines do not

limit personalization to a single genre but instead give the term "genre" a completely new meaning by creating a playlist of songs that are ostensibly unrelated yet that particular person finds to be excellent music. Using the aforementioned idea, Apple Music continuously optimizes its For You section.

Let's take a deeper look at one of the largest streaming music services that actively incorporate AI into its offerings. The first step to understanding its working algorithm is to understand its history. Without understanding the past, we cannot understand the present.

2.2 Basic Architecture of Deep Learning

DL is a cutting-edge technology based on neural networks that attempts to mimic how the human cortex operates. Today, we'd like to dig a little more into this topic. You should be aware that neural networks are far from relatively homogeneous. In reality, there are at least six types of neural networks and deep learning architectures based on them. we will discuss the most common and adaptable forms of deep learning architecture. RNN, CNN, and DSN will no longer be strange abbreviations.

To begin, we must clarify that deep learning system is made up of deep/neural networks with diverse topologies. The main premise is that neural networks are composed of numerous data-processing layers, including an input layer (raw data), hidden layers (which analyze and aggregate input data), and an output neurons (it produces the outcome: result, estimation, forecast, etc.). Neural networks have been constructed with a variety of layers that perform specific functions (which has made deep learning more feasible).

It's similar to a machine learning framework in that it helps you to make more practical use of this technology, speeds your work, and enables diverse undertakings without the need to create an ML algorithm from start.

There are several kinds of neural networks available for deep learning. These networks serve as the foundation for deep learning architectures. We can now identify six of the most typical deep learning architectures: **RNN, LSTM, GRU, CNN, DBN, DSN.**

However, we are not required to concentrate on all of the various neural networks and abbreviations in our thesis; our research focuses exclusively on RNN (Recurrent Neural Network).

2.2.1 Classification in Deep Learning

Deep Learning techniques are classified into two main learning methods:

- supervised learning
- unsupervised learning

as shown below in figure (2.4). Supervised learnings are further classified into **Regression** and **Classification** categories. and unsupervised learning technique is further divided into two more categories namely **Clustering** and **Association**. Our main focus will be on the unsupervised learning methods in which as I mentioned before that we are carrying out the clustering methods for our thesis work.

2.3 What are Neural Networks?

We need to understand what neural networks are before we can understand the neural networks discussed in section (2.2).

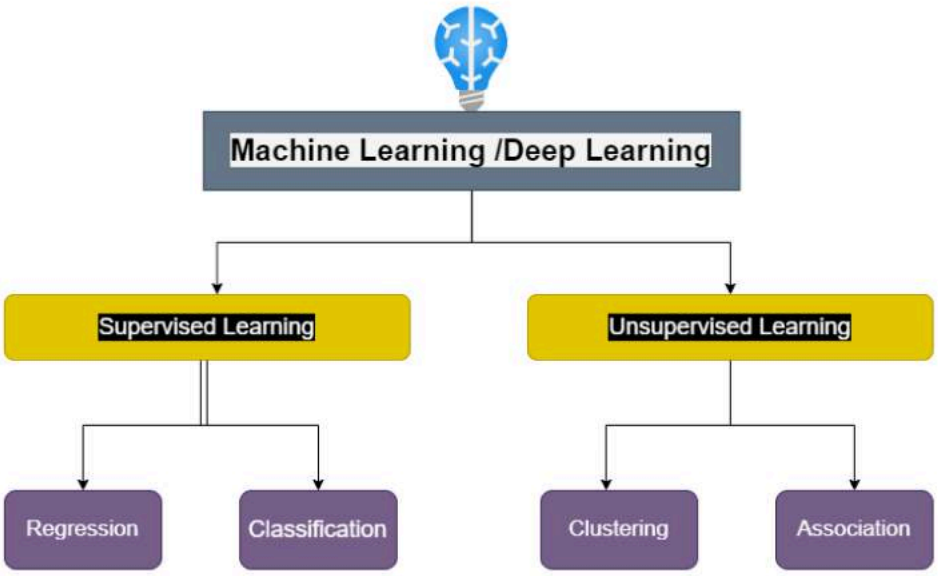


Figure 2.3 Hierarchy of Deep Learning

The human brain is thought to be among the most complex item in the universe. This hypothesis is primarily based on the neural network of the brain (Neurons), or how our organic nervous system processes information. The neural network is made up of a large number of highly linked functioning Neurons that work together to solve a specific problem. Breakthroughs in difficult machine-learning tasks are also attributed to neural networks.

A Neuron is the basic unit of computation in a neural network. Neurons absorb information, process it through numerous hidden layers of Neurons, and create output through the output layer.

Neurons

It is the neurons that form the brain and the nervous system that are the basic building blocks. During development, dendrites receive sensory input from the environment. By analyzing the sensory input, Axon terminals send it to the brain.

The general model of the neural network in machine learning is inspired by biological neurons. This model is known as a Perceptron. Though we don't need to study biology, these

are some of the concepts that function in deep learning too as the human nervous system works.

Perceptrons

A single-layer neural network with a perceptron has a single output. A Perceptron model is shown in the illustration below. Perceptron is the basic component in deep learning architecture.

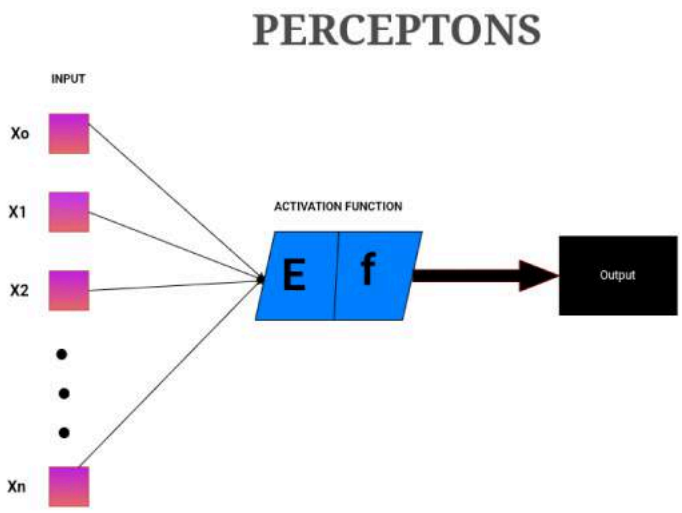


Figure 2.4 Perceptron neural network

Perceptron, x0, x1, x2 ...xn represents numerous inputs in the above diagram (independent variables). These inputs are each multiplied by weights, which are denoted by the letters y0, y1, y2,.... yn etc.

The Perceptron produces an output by summing the products of the inputs and weights, which are then supplied to an activation function.

Mathematically perceptrons can be represented as,

$$y^{\wedge} = g(\theta_0 + \sum_{i=1}^{m} x_i \theta_i)$$

Where, y is output, g is non linear function, theta(0) is bias and x(I) theta(0) is linear combination of inputs ,x represents input nodes, and i denotes the number of input nodes and biases ranging from i=1 to m.

2.3.1 What are Recurrent Neural Networks?

An artificial neural network that employs sequential data or time series data is called a recurrent neural network (RNN). They are implemented into well-known programs like Siri, voice search, and Google Translate. These deep learning algorithms are frequently employed for ordinal or temporal issues, such as language translation, natural language processing (nlp), speech recognition, picture captioning, and nlp. Recurrent neural networks (RNNs) use training data to learn, similar to feedforward and convolutional neural networks (CNNs). Because they use data from earlier inputs to affect the present input and output, they are characterized by their memory. The output of recurrent neural networks depends on the previous parts in the sequence, in contrast to typical deep neural networks, which presume that inputs and outputs are independent of one another.

RNN operates on the tenet that each layer's output is saved and fed back into the system's input in order to forecast that layer's output.

How to change a feed-forward neural network into a recurrent neural network is described below:

The nodes from multiple layers of the neural network are compressed to form a single layer of recurrent neural networks. The parameters of the network are A, B, and C.

Here The nodes from multiple layers of the neural network are compressed to form a single layer of recurrent neural networks. The parameters of the network are A, B, and C.

In this case, the input layer is "x," the hidden layer is "h," and the output layer is "y." The network parameters A, B, and C are used to enhance the model's output. The current input

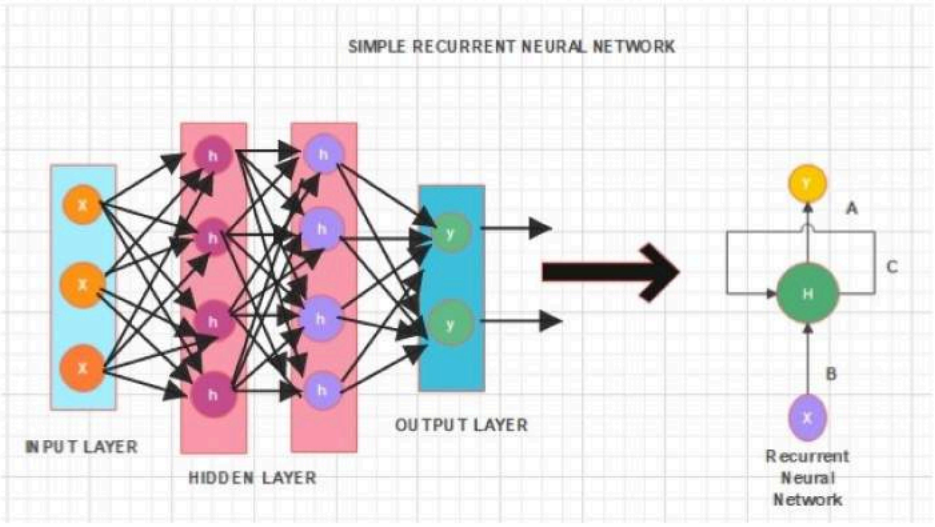


Figure 2.5 Simple Recurrent Neural Network

is a mixture of the input at x(t) and x at any given time t. (t-1). To enhance the output, the output at any given time is fetched back to the network.

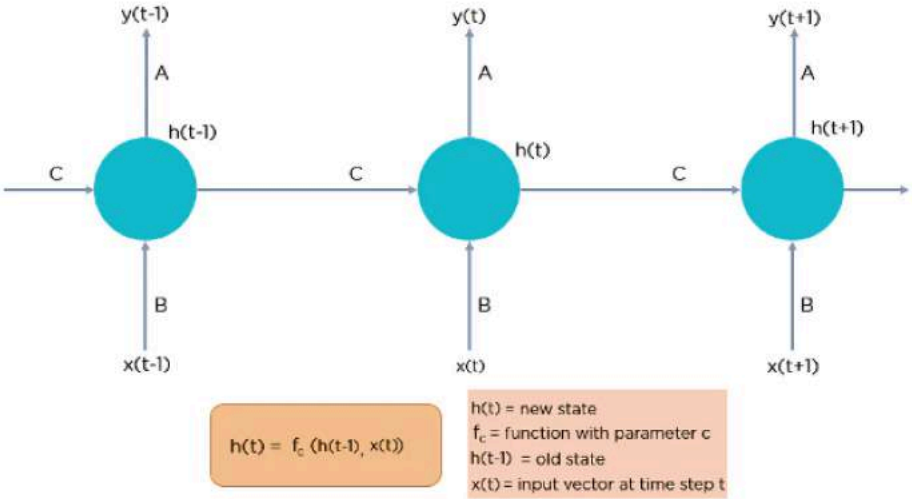


Figure 2.6 Fully connected Recurrent Neural Network

2.3.2 How Does Recuurent Neural Network Work?

The neural network's input is received by the input layer "x," which processes it before sending it to the middle layer.

Multiple hidden layers with unique activation functions, weights, and biases may make up the middle layer "h." Recurrent neural networks can be used with neural networks without memory, meaning that the various parameters of different hidden layers are not influenced by the previous layer.

So that each hidden layer has the same characteristics, the recurrent neural network will standardize the various activation functions, weights, and biases. Then, it will build one hidden layer and loop over it as many times as necessary rather than several hidden layers.

We've all used software that can convert spoken words into written ones (Apple Siri), or it can translate natural language (Google Translate). At first, you may have wondered how it did it.

There has been a significant advancement in the research underlying these systems over the past few years. For instance, Google developed a new system for Google Translate in late 2016 that makes use of cutting-edge machine learning methods. You can examine the improvement for yourself. Baidu's most recent text to speech feature is yet another astounding illustration. These techniques deal with the data which are sequential data to make predictions. On the other hand, recurrent (feedback) neural network is required to handle sequential data successfully. It has the capacity to "memorize" portions of the inputs and employ them in precise prediction. At the core of speech recognition, translation, and other technologies are these networks. So let's get started with a more thorough explanation.

The following actions are required steps for training a neural network:

- Include a dataset example.
- The network will use variables with randomly initialized values to perform some complicated computations on that example (called weights and biases).

- The outcome will be as expected.
- We will encounter an error when we compare that result to the intended value.
- The variables will be adjusted by propagating the error back through the same path.
- Repeat steps 1 through 5 until you are satisfied that your variables are well-defined. Additionally, prediction is achieved by using these variables to a fresh, unknown input.

Even though that explanation of a neural network is quite simplistic, it provides a good overview and can be helpful for someone who is completely unfamiliar with the subject.

lets examine with the above figure 2.6.

The input words x(t-1), x(t), and x(t+1) represent words from the text while the predicted words y(t-1), y(t), and y(t+1) represent words from the text. Information about previous input words is stored in h(t-1), h(t), and h(t+1).

We must encode the words into vectors because a neural network cannot process plain text. Although word embeddings (word2vec or GloVe) are the best method, we'll employ word2Vec or Doc2Vec for the purposes of this document of Genres,languages and lyrics. These are (V,1) vectors, where everything but the value at the i-th point are zero (V being the total number of words in our lexicon). For instance, if the word is banana and our vocabulary includes apples, apricots, bananas, kings, and zebras, the vector would be [0, 0, 1,..., 0,..., 0].

We utilize word embeddings since the vocabulary often only contains English terms.

Let's look at the equation below.

1. contains details about the words that came before it in the sequence. As you can see, h(t) is computed using the current word vector x(t+1) and the prior vector h(t+1). Additionally, we add a non-linear activation function (often a tanh or sigmoid function) to the final sum. Assuming that h 0 is a vector of zeros is acceptable.

$$1) \quad h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

$$2) \quad y_t = softmax(W^{(S)}h_t)$$

$$3) \quad J^{(t)}(\theta) = \sum_{i=1}^{|V|} (y_{t_i}^{'} log y_{t_i})$$

Figure 2.7 Equation of RNN

2. At a specific time step t, calculates the expected word vector. We create a (V,1) vector using the softmax function, whose elements all add up to 1. This probability distribution provides the index of the following word from the lexicon that is most likely to occur.

3. determines the error between the expected and real word at each time step t using the cross-entropy loss function.

The weights of the network at each stage are represented by these Ws, in case you're wondering what they stand for. The weights are matrices that, already established, are initialized with random elements and are then updated using the error from the loss function. By updating the weights, the back-propagation algorithm is used for this modifying.

Since RNN is the only topic here, we shall examine NLP and other ideas that are employed in this thesis work in the following section (section 2).

Part II

Section Two

Chapter 3

Natural Language Processing

3.1 Natural Langauge Processing

The field of computer science known as "Natural language processing" (NLP) is more particularly the field of "artificial intelligence" (AI) that focuses on providing computers the capacity to comprehend written and spoken words in a manner similar to that of humans.

Statistical, machine learning, and deep learning models are all combined in computational linguistics, which models human language using rules. Together, these technologies provide computers the ability to comprehend human language in the form of text or speech data and to fully "understand" its meaning, including the speaker's or writer's intention and sentiment.

Using Natural Language Processing (NLP), computers can translate text from one language to another, respond to voice commands, and summarize large amounts of text quickly - even in real time.

You've probably used natural language processing (NLP) in the form of voice-activated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. However, NLP is also taking on a bigger part in corporate solutions that assist reorganize workplaces, boost worker output, and make complex but vital company procedures simpler.

3.1.1 Tasks that are used in NLP

Because of the ambiguities in human language, developing software that correctly determines the intended meaning of text or speech input is exceedingly challenging. Homonyms, homophones, sarcasm, idioms, metaphors, grammar and usage exceptions, sentence structure variations—these are just a few of the human language irregularities that take humans years to learn but that programmers must teach natural language-driven applications to recognize and understand correctly from the beginning if those applications are to be useful.

Several NLP tasks dissect human text and audio data to help the computer understand what it is reading. Among these assignments are the following:

- **Speech Recognition:** The process of accurately translating spoken input into text is known as voice recognition, sometimes known as speech-to-text. Speech recognition is necessary for any software that responds to spoken instructions or questions. Due to the way humans communicate—they talk quickly, jumble words, use a range of accents and intonations, and commonly employ poor grammar—speech detection is incredibly difficult.

- **Speech Tagging:** The procedure of identifying the part of speech of a certain word or passage of text based on its use and context is known as grammatical tagging. In the sentences "I can create a paper aircraft" and "What kind of automobile do you own?," the word "create" is classified as a verb and a noun, respectively.

- **Named entity recognition:** Words or phrases are recognized as helpful things by NEM. NEM identifies "Kentucky" as a place or "Fred" as the name of a guy.

- **Sentiment analysis:** looks for intangible elements in text, such as attitudes, feelings, sarcasm, bewilderment, and mistrust.

- **Natural Language Generation:** the process of transforming structured data into human language; sometimes known as speech-to-text or voice recognition in reverse.

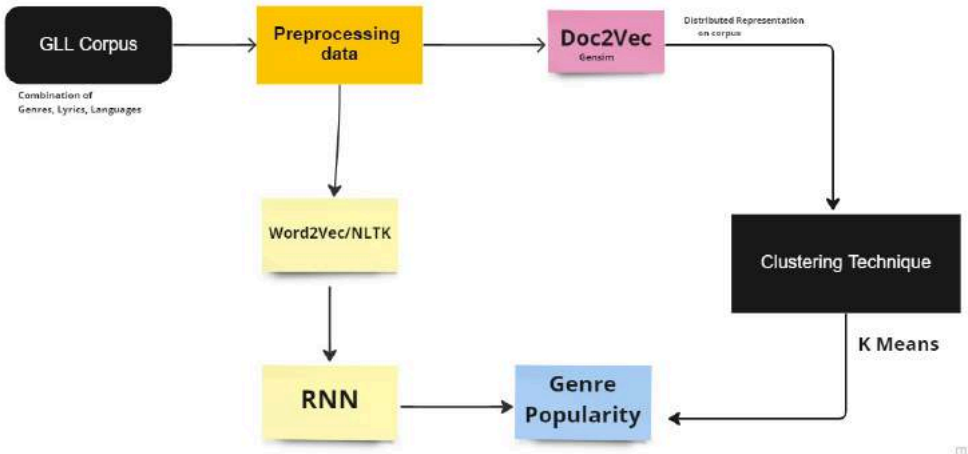
3.1.2 Lyrics and Genre Prediction

The study on music mining has been fuelled by the recent, substantial increase in streaming music consumption. More than 80 percent of those who subscribe to internet music do lyrics searches. It shows that words have a significant role in the musical experience. The realization that lyrics have not yet been fully utilized for computationally analyzing music and language serves as the inspiration for this effort. A song may be experienced primarily in three ways: visually through video, auditorily through music, and linguistically through words. Lyrics offer two key benefits over visual and audio elements when it comes to music analysis. First, the words primarily communicate the song's goal. Second, computational analysis of lyrics as text data requires far fewer resources. We concentrate on lyrics in this research to show their utility for two large fields: music mining and NLP.

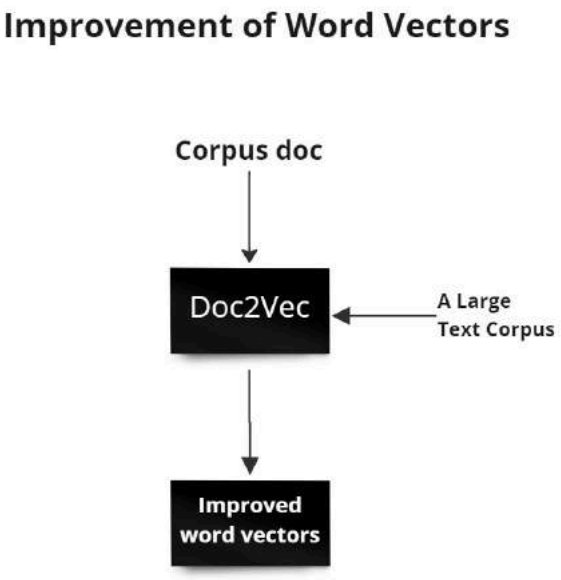
Moving away from handmade properties and toward distributed representation is a current trend in NLP. Many NLP tasks have been successfully completed using techniques like word2vec and doc2vec in combination with deep learning. We concentrate on unsupervised learning problems, notably clustering methods: genre, languages, and lyrics, given lists of song data. To jointly learn the representation of lyrics as well as genre and language labels, we employ distributed representation learning techniques. We test multiple conventional unsupervised machine learning and Deep Learning models using these learnt vectors. Please see Figures 3.1 and 3.2 for a summary of our methodology.

As shown above, our methodology works, and we will learn more about coding based on this methodology in the next chapter 4.

As a result of our work, we have provided three new findings. The strength of distributed representation of lyrics for music mining and NLP tasks is first shown in this paper, which is the first to do so. Additionally, we demonstrate that lyrics by themselves can serve as reliable



Genre Prediction based on lyrics



Vectorization of Words

indications of genre suggestion based on language, contrary to previous research. The third advantage is the possibility of improving the quality of words vectors by utilizing the data from songs.

3.2 Dataset

More than 18k songs and other facts, including artists, albums, audio features, lyrics, and languages of the lyrics, genres, and subgenres, are included in this dataset.

According to my professor, I am utilizing this open-source dataset from Kaggle for my research. The size of this dataset is 44.1 MB.

These collections of tracks include music from six genres:

1. Rap.
2. Rock.
3. EDM.
4. RB.
5. Latin.
6. Pop.

Rap was one of the genres that were easiest to categorize, partly because of the speechiness characteristic. Rock songs were easier to distinguish from dance tracks because to low danceability, and EDM songs were more accessible to identify according to the fast pace.The hardest genres to categorize were RB, pop, and latin music; nonetheless, R and B songs tended to be longer in length, while latin songs were slightly more danceable than pop recordings. There are some sub genres aswell (eg: classic Rock,hiphop,new jack

swing,hard rock),basically these are part of these main genres.so we will analyze them aswell to understand more about the dataset.

These are all the specifics about the dataset that I am utilizing for my research project, so let's play with the data in the technique section and see how we can use the genres that I described before to forecast genres and calculate the distance between them as our major outputs.

Data Loading and Description

```
[8]: songs_df = songs_df.drop(['track_id','track_album_id','playlist_id','track_album_release_date'] ,axis=1)
     songs_df.head(5)
```

:[8]:

	track_name	track_artist	lyrics	track_popularity	track_album_name	playlist_name	playlist_genre	playlist_subgenre	danceability	energy	...	loudness
0	Pangarap	Barbie's Cradle	Minsan pa Nang ako'y napalingon Hindi ko alam ...	41	Trip	Pinoy Classic Rock	rock	classic rock	0.682	0.401	...	-10.068
1	I Feel Alive	Steady Rollin	The trees, are singing in the wind The sky blu...	28	Love & Loss	Hard Rock Workout	rock	hard rock	0.303	0.880	...	-4.739
2	Poison	Bell Biv DeVoe	NA Yeah, Spyderman and Freeze in full effect U...	0	Gold	Back in the day - R&B, New Jack Swing, Swingbe...	r&b	new jack swing	0.845	0.652	...	-7.504
3	Baby It's Cold Outside (feat. Christina Aguilera)	CeeLo Green	I really can't stay Baby it's cold outside I'v...	41	CeeLo's Magic Moment	Christmas Soul	r&b	neo soul	0.425	0.378	...	-5.819
4	Dumb Litty	KARD	Get up out of my business You don't keep me fr...	65	KARD 2nd Digital Single 'Dumb Litty'	K-Party Dance Mix	pop	dance pop	0.760	0.887	...	-1.993

5 rows × 21 columns

Figure 3.1 Data Format

The Above figure shows what the format of data looks like with its column names. However based on the above variables there are several unsupervised learning techniques are used to perform on the dataset to get the information about the songs and genres prediction, but in out case, we are using unsupervised learning techniques to analyze the data with clustering techniques.

3.3 Word embeddings

Simply said, a word embedding is a different way to describe text where words are represented as real-valued vectors rather than a string of letters. This representation is "learned" in a way that similarly spelled words also have similarly spelled vector representations.

One of the most used methods in natural language processing is word embedding (NLP). It is frequently claimed that word embeddings are essential to the functionality and effectiveness of SOTA models. Word embeddings play a crucial role in the rapid evolution of language models including RNNs, LSTMs, ELMo, BERT, AlBERT, GPT-2, and the most recent GPT-3. For Example Things have recently changed for chat GPT, as it has increased its subscriber base in just five days. In addition to this, it also uses NLP technology so now we understand how embedding techniques perform in AI.

3.3.1 Working of Word embeddings?

Word embeddings allow words and complete sentences to be numerically represented. We strive to encrypt words in a sentence to numbers so that the computer can read it and process it since we know that computers can communicate with each other in the language of numbers. Additionally, we want computers to establish connections between the words in a sentence or document and other terms in the same.

We want word embeddings to capture both the semantic and syntactic similarities of the same words as well as the context of the paragraph or prior phrases.

For instance, if we take an example of a sentence:

The cat is playing and the dog was sleeping.

Let's take two subjects here as cat and dog, and then switch this sentence around.

The dog is playing and the cat was sleeping

Both sentences above maintain a semantic relationship, since cats and dogs are animals, and the sentence makes sense. Due to its proper grammar, the same sentence is also a syntactic relationship.

we need to achieve this kind of semantic and syntactic relationship is been used to map the documents or paragraphs to a number.

numbers are nothing but the vectors of words. They should be learnable vectors that the computer understands.

let us understand how it represents mathematically :

$$f\theta(Wn) = \theta n$$

where W is the word sentence,

for instance, the word cat and dog can be represented as :

W(cat) = (0.9, 0.2, 0.1, -0.223...)

W(dog) = (0.34, 0.56, 0.64, -0.32...)

if the model is able to find similarity between words then both words will be in a vector space as listed above.

so till now, we have seen embeddings of just two words "Cat and Dog, so what if we have more words in the sentence? the main job of the word embedding model is to cluster similar words and similar information based on their relationship.

In our work we are no working on cats and dogs though, our model will work with a large dataset and find out the vectors and clusters according to them.

We can see below a sample of one of our results, which shows how clustering works for lyrical words based on vectors.

Our data sets have been clustered by word2vec based on similarities between genres in the diagram.

We use neural networks to embed words into our clusters because they are effective methods for clustering. What are the benefits of using neural networks for word embeddings?

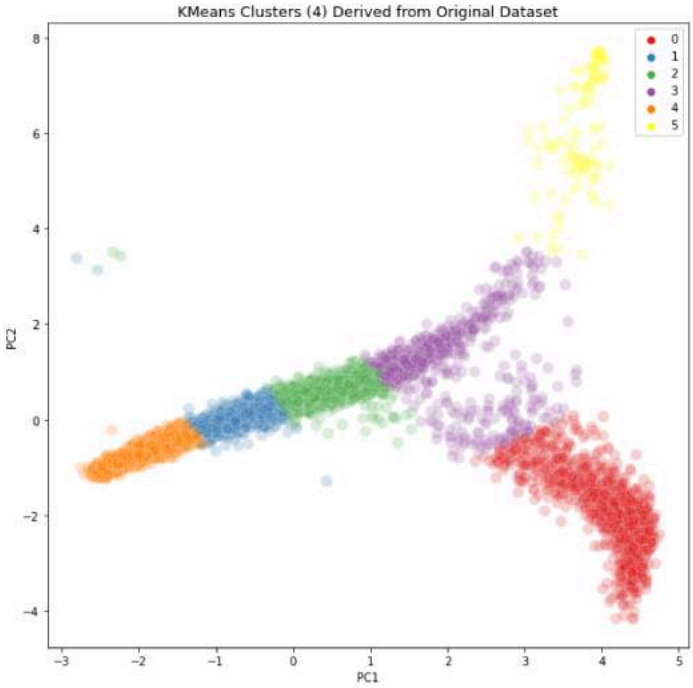


Figure 3.2 Clustering of Genres

One of the most widespread fallacies is that deep neural networks are necessary for word embeddings. You'll notice that when we develop various word embedding models, the models for all of the embeddings are deep neural networks, with some also being linear models.

We employ neural networks to build word embeddings for the following reasons:

It helps in locating the embedding space's closest neighbors. Unsupervised learning tasks may use it as input or dimensionality reduction techniques may use it to reduce the size of the input. Words are converted into a vector of continuous variables.

3.4 Introduction to Word2Vec

Word2vec, a two-layer neural network, uses the word "vectorization" to analyze text. A text corpus serves as its input, and its output is a collection of vectors (numbers), which stand in for the words in the input corpus. Word2vec converts text into a numerical form that deep neural networks can understand.

The goal of Word2vec is to bring together in vectorspace the vectors of words that are close in meaning. In other words, it looks for mathematical similarities. Word characteristics, such as the context of individual words, are dispersed numerically represented as vectors via Word2vec. Without the assistance of anyone.Word2vec can create remarkably accurate assumptions about a word's meaning based on prior occurrences if given sufficient data, use, and circumstances.

These inferences can be used to group documents and categorize them according to subject or to determine a word's relationship with other words. Both syntactic and semantic similarities between the words are captured by the Word2vec model. One of the well-known instances of the word2vec vectors taught with vector algebra is mentioned in below example : (for example, "king" is to "man" as "queen" is to "woman").

How Word2vec model is useful in Recommender system?

so in the next section let us understand how the word2vec model can be used in music recommendation systems. and let us understand how each of the songs is similar based on genres and languages and also lyrics with the vector space.

The process is carried out by merging numerous recommender systems, as you can see in the above image which shows how my Spotify tracks have been arranged into these tiles

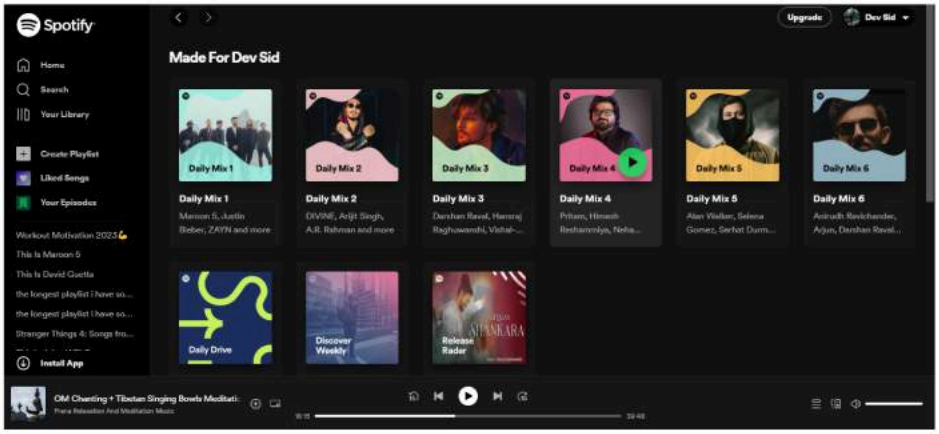


Figure 3.3 Made for you spotify's Recommender Algorithm tiles on spotify

depending on my history, search, and playlist likes. This allows the user to revel in his or her musical preferences. Users might not have enough time to manually compile a playlist by listening to every song that is offered. In its place, a recommender system is built to make it simple for them to locate appropriate music fast. The Spotify "Made For You" feature is one illustration that you may have already seen. In this study, we will show how to produce song recommendations using a neural network technique, especially the Word2Vec model, and how to extract song embeddings.

In reality, the word embedding technique may be used to various types of items, linking every product on an e-commerce website, any YouTube video, or any Netflix film to a vector. Of course, music may also be a vector in this situation.

As shown in figure 3.6, word pairs are taken from the lyrics of one of our songs and transcribed, and highlighted words are used as inputs. Using this method, we are performing a network task. We will be using the network to determine the probability of nearby words that have been chosen.

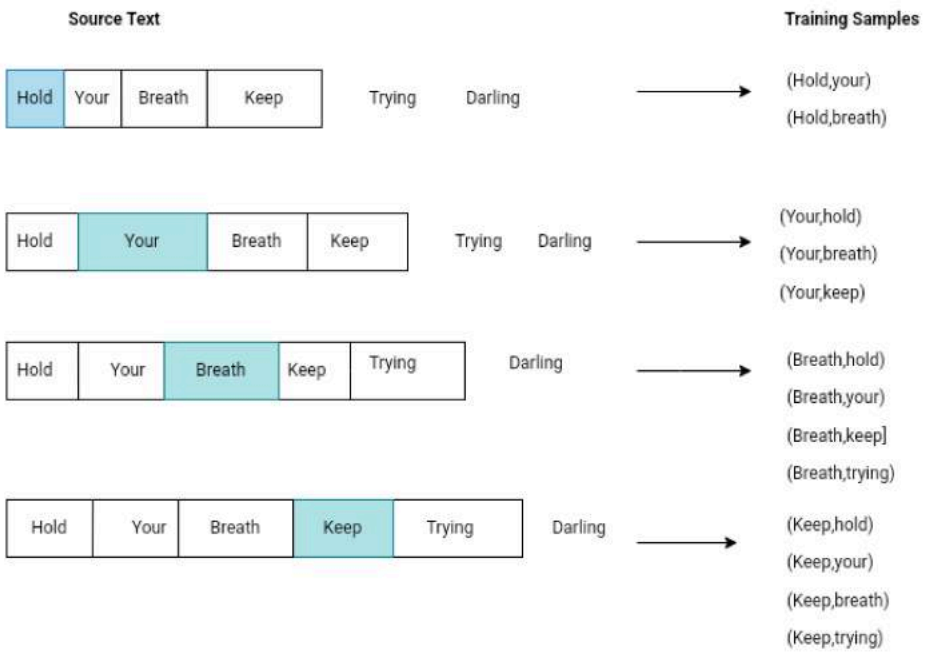


Figure 3.4 Word pairs taken from lyrics using a window size of 2. the words highlighted are considered to be input words.

3.5 Document Embeddings

Another way to think of document embedding is as a continuous calculation method for word embeddings. We have trouble establishing the contextual link between the words in vector form since word embedding estimates or learning transforms the entire corpus into vectors. The possibility of determining the contextual associations between words is made possible by extracting tiny corpora and turning them into vector representations.

Acronyms, for instance, might have various contexts in several paragraphs, making it challenging to distinguish between many meanings using a simple vector representation. Our vector representations will be more meaningful if we discrete the entire corpus sentence or sentence-by-sentence before producing vector representations.

So in our Thesis Research, we are carrying out the embeddings of the document as a com-

bination of Lyrics and genres, and languages and then finding out the vectors space on the document named as("TaggedDocument").

3.5.1 What are Doc2Vec Models?

Natural language processing uses a variety of methods, such as Word2Vec and a bag of words, to encode text input as a vector for analysis. The goal of these methods is to represent documents as fixed-length vectors. Some drawbacks of encoding text as a fixed-length vector can include the loss of semantic connection information between words by the models. These models, for instance, are unable to depict the phrase "powerful is closer to strong."

The Doc2Vec and Word2Vec models are pretty similar. We have detailed information regarding Word2Vec models in one of our articles. To summarize this paper, word2vec models may be used to extract word embeddings from the whole corpus. While Doc2Vec suggests a technique for extracting word embedding from corpus paragraphs. Instead of being vector representations of the entire corpus, we can alternatively think of these word vectors as vector representations of paragraphs.

Researchers have often utilized unsupervised learning techniques to learn continuous distributed vector representations while studying Doc2Vec. We might think of Doc2Vec modelling as a technique that applies vectorization to short bits of text documents, such as phrases or sentences while learning to represent text as vectors. This technique may be used to anticipate words in paragraphs. These models are capable of predicting the word in a particular context by concatenating the paragraph with a number of word vectors from the paragraph.

We may infer that learning with paragraph vectors is derived from word vector learning. Even if we can predict the next word in a phrase using a word vector generated by random factorization, we can also predict the next word using a paragraph vector, which can provide

superior results since it can also contain information about the semantic relationship.

In a paragraph vector-matrix, the following elements can be found:

1. Every paragraph is mapped to a different vector in a process known as **paragraph vector representation.**
2. Every word from a paragraph is represented as a separate vector in a **word vector.**

The learning process utilizing a paragraph vector can be shown by the diagram below.

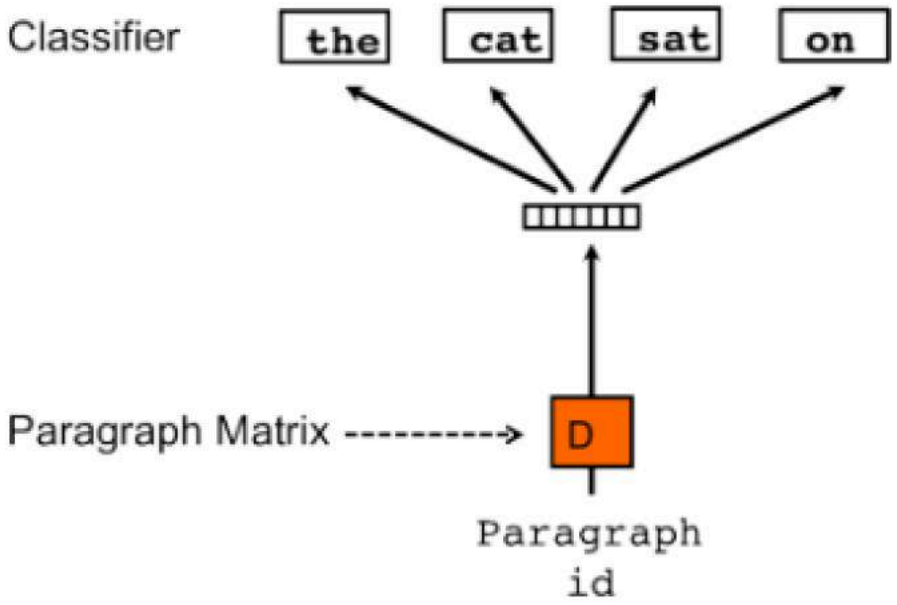


Figure 3.5 DBOW Working Model

Additionally, Word2Vec-like models for Doc2Vec exist in two variations:

- Distributed memory model.
- Distributed Bag of words.

These variations are comparable to Word2Vec variations. The continuous bag of words model and the distributed bag of words model have similarities with the distributed memory model and the skip-gram model, respectively. The distributed bag of words model is the main topic of this essay. Let's go on to the introduction of the distributed bag of words model without spending any more time.

Doc2Vec differs from word2Vec in that it uses the paragraph id as a distinctive vector. This vector may be thought of as an additional word that serves as a memory for the process. The memory algorithm recalls the words' present context and guesses what is lacking based on that context.

Using the Gensim package, the Doc2Vec model may be implemented. In this post, we'll examine how Doc2Vec is implemented, but first, it's important to understand how the Gensim library works because it may also be used to construct other models.

3.5.2 Gensim Library

Gensim is an open-source Python text processing package. It primarily deals with the representation of text texts as semantic vectors. Gensim is an acronym for creating similar. An in-depth analysis of the architecture for text processing reveals that this library makes use of unsupervised machine learning methods. We can automate the process of determining the semantic structure of text data by using Gensim's algorithms. It basically looks at the data corpus' statistical co-occurrence pattern. Most of the time, we don't need any human involvement because it uses unsupervised learning algorithms.Gensim is used in Natural Language Processing as package for Topic Modelling for humans where Human intervention is zero.

It is an excellent toolkit for handling texts, utilizing word vector models (like Word2Vec, FastText, etc.), and creating subject models.

Another important benefit of using gensim is that you can handle big text files without having to load the full file into memory.

This Library supports all the python versions with good condition.we can install this library with the following code that i have installed in my final development model.

! pip install –upgrade Gensim

First, we need to know the implementation of gensim library in our work.

3.5.3 Implementing Doc2vec using Gensim using DBOW Method

The Gensim package makes it simple to implement the Doc2Vec model, as was explained in the paragraph above. Following the above-mentioned installation processes, we are now prepared to utilize this library. First, let's import the library.

```
10 from gensim.models import Doc2Vec
11 import gensim
12 from gensim.models.doc2vec import TaggedDocument
13
```

We already have a dataset of Spotify songs we mentioned in the dataset section.lets use that dataset to create a (Tagged Document) class and train that particular document to see results. as you can see in below screenshot.

Here is the initial list of terms that have been tagged. This list may also serve as the basis for our opening paragraph vector. We must right away instantiate the Doc2Vec model.

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [30]:

```
train, test = train_test_split(df, test_size=0.3, random_state=42)
import nltk
from nltk.corpus import stopwords
def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            if len(word) < 2:
                continue
            tokens.append(word.lower())
    return tokens

train_tagged = train.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['lyrics']), tags=[r.playlist_genre]), axis=1)
test_tagged = test.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['lyrics']), tags=[r.playlist_genre]), axis=1)
```

In [36]:

```
train_tagged.values[2]
```

Out[36]:

```
TaggedDocument(words=['pop-i', 'woke', 'up', 'it', 'was', 'waited', "'til", '11', 'just', 'to', 'figure', 'out', 'that', 'no',
'one', 'would', 'call', 'think', "'ve", 'got', 'lot', 'of', 'friends', 'but', 'do', "n't", 'hear', 'from', 'them', 'what',
"'s", 'another', 'night', 'all', 'alone', 'when', 'you', "'re", 'spending', 'everyday', 'on', 'your', 'own', 'and', 'here', 'i
t', 'goes', "'m", 'just', 'kid', 'and', 'life', 'is', 'nightmare', "'m", 'just', 'kid', 'know', 'that', 'it', "'s", 'not', 'fai
r', 'nobody', 'cares', "'cause", "'m", 'alone', 'and', 'the', 'world', 'is', 'having', 'more', 'fun', 'than', 'me', 'tonight',
'and', 'maybe', 'when', 'the', 'night', 'is', 'dead', "'ll", 'crawl', 'into', 'my', 'bed', "'m", 'staring', 'at', 'these', 'fou
r', 'walls', 'again', "'ll", 'try', 'to', 'think', 'about', 'the', 'last', 'time', 'had', 'good', 'time', 'everyone', "'s", 'go
t', 'somewhere', 'to', 'go', 'and', 'they', "'re", 'gon', 'na', 'leave', 'me', 'here', 'on', 'my', 'own', 'and', 'here', 'it',
'goes', "'m", 'just', 'kid', 'and', 'life', 'is', 'nightmare', "'m", 'just', 'kid', 'know', 'that', 'it', "'s", 'not', 'fair',
'nobody', 'cares', "'cause", "'m", 'alone', 'and', 'the', 'world', 'is', 'having', 'more', 'fun', 'than', 'me', 'what', 'the',
'hell', 'is', 'wrong', 'with', 'me', 'do', "n't", 'fit', 'in', 'with', 'anybody', 'how', 'did', 'this', 'happen', 'to', 'me',
'wide', 'awake', "'m", 'bored', 'and', 'ca', "n't", 'fall', 'asleep', 'and', 'every', 'night', 'is', 'the', 'worst', 'night',
'ever', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid', 'yeah', "'m", 'just', 'kid', "'m",
'just', 'kid', "'m", 'just', 'kid', 'and', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid',
'and', 'life', 'is', 'nightmare', "'m", 'just', 'kid', 'know', 'that', 'it', "'s", 'not', 'fair', 'nobody', 'cares', "'cause",
"'m", 'alone', 'and', 'the', 'world', 'is', 'nobody', 'wants', 'to', 'be', 'alone', 'in', 'the', 'world', "'m", 'just', 'kid',
'and', 'life', 'is', 'nightmare', "'m", 'just', 'kid', 'know', 'that', 'it', "'s", 'not', 'fair', 'nobody', 'cares', "'cause",
"'m", 'alone', 'and', 'the', 'world', 'is', 'nobody', 'wants', 'to', 'be', 'alone', 'in', 'the', 'world', 'nobody', 'cares',
"'cause", "'m", 'alone', 'and', 'the', 'world', 'is', 'having', 'more', 'fun', 'than', 'me', 'tonight', "'m", 'all', 'alone',
'tonight', 'nobody', 'cares', 'tonight', "'cause", "'m", 'just', 'kid', 'tonight-en'], tags=['pop'])
```

The below line of code is used to build the Doc2vec model.

Document embeddings with Doc2Vec (extension of Word2vec)

In [40]:
```
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0, min_count=2, sample = 0, workers=cores)
model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%| 12735/12735 [00:00<00:00, 304054.45it/s]
```

In [41]:
```
%%time
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(train_tagged.values)]), total_examples=len(train_tagged.values), epochs=
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha
```

```
100%| 12735/12735 [00:00<00:00, 212853.26it/s]
100%| 12735/12735 [00:00<00:00, 692857.48it/s]
100%| 12735/12735 [00:00<00:00, 807609.15it/s]
100%| 12735/12735 [00:00<00:00, 1259240.45it/s]
100%| 12735/12735 [00:00<00:00, 709902.20it/s]
100%| 12735/12735 [00:00<00:00, 814592.53it/s]
100%| 12735/12735 [00:00<00:00, 1054246.66it/s]
100%| 12735/12735 [00:00<00:00, 1264846.35it/s]
100%| 12735/12735 [00:00<00:00, 703191.96it/s]
100%| 12735/12735 [00:00<00:00, 982407.19it/s]
100%| 12735/12735 [00:00<00:00, 770904.94it/s]
100%| 12735/12735 [00:00<00:00, 798207.68it/s]
100%| 12735/12735 [00:00<00:00, 608101.98it/s]
100%| 12735/12735 [00:00<00:00, 751194.86it/s]
100%| 12735/12735 [00:00<00:00, 425555.59it/s]
100%| 12735/12735 [00:00<00:00, 798136.12it/s]
100%| 12735/12735 [00:00<00:00, 751205.42it/s]
100%| 12735/12735 [00:00<00:00, 798136.12it/s]
100%| 12735/12735 [00:00<00:00, 751173.73it/s]
100%| 12735/12735 [00:00<00:00, 608136.59it/s]
100%| 12735/12735 [00:00<00:00, 751120.91it/s]
100%| 12735/12735 [00:00<00:00, 798148.04it/s]
100%| 12735/12735 [00:00<00:00, 798159.97it/s]
100%| 12735/12735 [00:00<00:00, 798183.82it/s]
100%| 12735/12735 [00:00<00:00, 709694.69it/s]
100%| 12735/12735 [00:00<00:00, 709543.86it/s]
100%| 12735/12735 [00:00<00:00, 851375.72it/s]
100%| 12735/12735 [00:00<00:00, 751152.60it/s]
100%| 12735/12735 [00:00<00:00, 216439.53it/s]
100%| 12735/12735 [00:00<00:00, 798195.75it/s]
CPU times: total: 10min 51s
Wall time: 4min 14s
```

Let's come to a conclusion from the model using numerical data.

The Doc2Vec model and Gensim library have been covered. Along with this introduction, we have shown how to use the Gensim library to quickly and simply implement the Doc2Vec model. Since the library's emphasis is on practical issues.

```
In [42]:
def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors

def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors
```

```
In [43]:
y_train, X_train = vec_for_learning(model_dbow, train_tagged)
y_test, X_test = vec_for_learning(model_dbow, test_tagged)
```

```
In [44]:
X_train[0]
```

```
Out[44]: array([-0.19294877, -0.2742632 ,  0.6467644 , -0.25354752,  0.36493817,
                -0.40804225, -0.29514885, -0.2611752 ,  0.13391656,  0.5535047 ,
                 0.21244995, -0.25543854,  0.46290022, -0.72762376, -0.33906227,
                 0.50380254, -0.31047982,  0.6312567 , -0.34977335,  0.30268738,
                -0.6240096 ,  0.00452384, -0.64921045, -0.6674144 , -0.5715819 ,
                 0.19219248,  0.15305632,  0.1717297 ,  0.35616493,  0.29738826,
                 0.21059397, -0.00488677, -0.4605665 , -0.7592948 ,  0.30794144,
                -0.56145096, -0.02599613,  0.24012025, -0.06423301,  0.38546798,
                -0.31480324, -0.34030503,  0.40097678, -0.5226217 ,  0.38020012,
                -0.3041114 ,  0.6565691 , -0.42317075, -0.15175667, -0.70886004,
                 0.20129465, -0.14487544,  0.5250421 , -0.24428184, -0.02871065,
                 0.7254043 ,  0.3670491 , -0.63664365,  0.5519919 ,  0.44797155,
                 0.0020396 , -0.01270282,  0.48947072,  0.5210162 , -0.06528765,
                -0.23543291,  0.03228668, -0.16077054,  0.29993045, -0.1560635 ,
                -0.16193886,  0.3334631 ,  0.66818464,  0.13535881,  0.00603267,
                -0.46563497, -0.3472587 ,  0.293802  , -0.37381053, -0.05052046,
                 0.38849762, -0.12683783,  0.03771146, -0.19183639, -0.02188664,
                -0.52972513, -0.1099641 , -0.02111275,  0.5320757 ,  0.12498055,
                -0.13927367,  0.13821092, -0.02372115, -0.47595865, -0.3182143 ,
                -0.01210294,  0.46794453, -0.17244206,  0.60132843,  0.13007602,
                 0.35667366,  0.4282921 , -0.25044677,  0.10058066,  0.47624382,
                -0.28431144, -0.18661247, -0.18084103, -0.10542763, -0.18163018,
                 0.1544591 , -0.14628677,  0.03487036, -0.36399308,  0.36433765,
                -0.12989481,  0.12044547, -0.5127751 ,  0.11863443, -0.19119853,
                -0.01060296,  0.06294227,  0.01017837,  0.6709838 ,  0.39714885,
```

We have also discussed about document embedding and discovered that to model document embeddings from word corpora, Doc2Vec models are necessary. We also spoke about how the distributed bag of words model is a particular kind of Doc2Vec model that may be utilized to do our NLP jobs more effectively.

3.6 Clustering Analysis

3.6.1 Introduction

Clustering is the process of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). It is a method of unsupervised learning, which means that the algorithm does not have any prior knowledge of the groupings in the data.
As we mentioned above, clustering is basically an unsupervised machine learning method, in which some references can be taken from the dataset, which consists of unlabelled input data. Thus, it is commonly used as a method for finding meaningful details by grouping data.

Our thesis work utilizes genre analysis in which data points are categorised by their specific features, and we are using this model to analyze genres and group them based on the results. The result is some insightful visual results.

It is natural for people to have a different kind of music taste based on their hobbies, lifestyles, and professions, so the music streaming industry has been working on these kinds of interests for the past five years, so it is difficult for these industries to recommend music to their audiences.

In this situation, we use cluster analysis to identify music genres. You're given a dataset with a number of popular Spotify songs and genres, as well as some lyrics, which include all audio characteristics of each song, artists, Spotify features, and the names of the songs. It is my aim to group music genres based on their similar audio characteristics and recommend them.

Clustering algorithms vary widely, each with its own advantages and disadvantages. Some popular algorithms include:

K-means: Kmeans clustering is one of the popular algorithms in unsupervised learning. It works by dividing the data into k clusters, where k is said to be a user-specified parameter like the user can mention how many clusters should be fixed for our working model. The algorithm works by randomly initializing k centroids, which is nothing but the center point of each cluster, and then repeatedly assigning the data point to the clusters whose centroid is closest to it until the assigned data stop changing its positions. so in detail, we will see this in the next sections.

Hierarchical Clustering : This algorithm does create clusters but in a tree format, where each node in a tree represents a cluster. There are two types of hierarchical clustering :

- Agglomerative (bottom-up)
- Divisive (top-down)

In **Agglomerative Clustering**, each of the data points starts with separate clusters and this algorithm repeatedly merges the closest clusters until all of the data points are in the closest clusters and until all the data points are in one cluster.In **divisive clustering**, all the data points start in one cluster and the algorithm iteratively splits the cluster into smaller clusters until each data point is in its own cluster.

DBSCAN: Density-Based Spatial Clustering of Applications with Noise, this algorithm creates clusters based on the density of points in a particular area. It does not require the user to specify the number of clusters in advance and is able to discover clusters of any shape.

Gaussian Mixture Model: This algorithm assumes that the data is generated by a mixture of different Gaussian distributions, and it estimates the parameters of the distributions to identify clusters.

Spectral Clustering: This algorithm makes use of the eigenvectors of the similarity matrix of the data to find clusters.

There are some factors that should be considered when choosing a clustering algorithm. Some of these include:

- The number of clusters: Some algorithms require the user to specify the number of clusters in advance, while others do not.
- The shape of the clusters: Some algorithms are only able to identify clusters that are spherical in shape, while others are able to identify clusters of any shape.
- The size of the data: Some algorithms are only able to handle small or medium-sized datasets, while others can handle very large datasets.

It is also important to note that, Clustering is not a one-size-fits-all solution, and the choice of algorithm depends on the specific problem and dataset at hand. Therefore, it is always recommended to try out different algorithms and evaluate their performance using appropriate evaluation metrics.

As part of our thesis work, we are primarily using K means clustering, which is regarded as the best in many cases, and we will be implementing its method in the following section.

To evaluate the quality of a clustering, we use a variety of metrics such as:

Silhouette score: It ranges between -1 and 1, a higher value indicates that the data points in the cluster are more similar to each other and different from the data points in other clusters.

Calinski-Harabasz Index: It measures the ratio of the sum of between-cluster variance to the sum of within-cluster variance.

Davies-Bouldin Index: It measures the average similarity between each cluster and its most similar cluster.

In terms of visualization, Clustering results can be visualized in a variety of ways such as:

Scatter plots: where each data point is plotted as a point. The simple scatter plot, a close relative of the line plot, is another often-used plot style. Here, the points are each individually represented by a dot, circle, or another form rather than by line segments.

Since other clustering methods are not part of our thesis, let's not spend much time on them in the Upcoming section so we can focus on the methods we are using for our research.

3.7 K Means Clustering Analysis

A collection of n observations is divided into k groups by the K means clustering method. When you don't have existing group labels and you want to assign comparable data points to the number of groups you provide, use K means clustering (K).

In general, clustering is a technique that uses data patterns to assign comparable data points to groups. Algorithms for clustering identify comparable data points and group them into sets. One such technique is K means clustering. A cluster is a collection or set of related data points.

What "K represents" is as follows:

You indicate how many clusters (K). the method of allocating observations to the cluster with the closest center (mean).

The k-means algorithm will be implemented step-by-step in this section. Writing a Python k-means clustering pipeline requires a fundamental understanding of the algorithm's workings. I understand k-means and am the ideal candidate to address your clustering issue or analysis.

Before performing clustering analysis on data that we have we need to first understand about **elbow method** to find out the right number of clusters.

3.7.1 Elbow Method

It is the simplest and most often used unsupervised learning method of the iterative type. In this, the K number of centroids in the data are initialized at random (the number of k is determined using the Elbow technique), and these centroids are iterated until the centroid's position remains unchanged. For a better understanding, let's examine the procedures involved in K means clustering.

In the Elbow technique, the number of clusters (K) is truly variable and ranges from 1 to 10. We are computing WCSS for each value of K. (Within-Cluster Sum of Square).

The sum of the squared distances between each point and the cluster's centroid is known as WCSS. The plot of the WCSS with the K value resembles an elbow. The WCSS value will begin to drop as the number of clusters rises. The highest WCSS value is when K = 1. When we examine the graph, we can observe that it abruptly changes at one point, forming an elbow. The graph then begins to travel nearly parallel to the X-axis from this point on. The best K value, or the most clusters, is the one that corresponds to this location.

Now let us implement the K means clustering method for the analysis of our dataset and grouping them. below code blocks, you will see the implementation of kmeans.

Implementation of Kmeans

First of all, we have to import all the essential libraries to carry out the further processes. in the below code block you will find the libraries that we required.

```
In [25]:  1 from sklearn.cluster import KMeans
          2 model = KMeans(n_clusters = 6,init='k-means++',random_state=49)
          3 model.fit(X_train)

Out[25]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
                n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
                random_state=49, tol=0.0001, verbose=0)
```

```
labels = model.labels_
cluster_center=model.cluster_centers_
```

```
cluster_center
```

```
array([[ 0.51205407,  0.26649424,  0.90241309, ..., -0.09212171,
        -0.37043781,  0.57423351],
       [ 0.45112201,  0.27994382,  0.81651045, ..., -0.1418154 ,
        -0.36768555,  0.55361764],
       [ 0.4415233 ,  0.41966983,  0.88872495, ..., -0.38047016,
        -0.680373  ,  0.64575728],
       [ 0.06373378,  0.04069745,  0.14572571, ..., -0.07735449,
        -0.11231906,  0.10867206],
       [ 0.3268094 ,  0.2722219 ,  0.69805395, ..., -0.27002691,
        -0.41384042,  0.50126829],
       [ 0.54242459,  0.27594353,  0.96211601, ..., -0.06060533,
        -0.42206391,  0.58879165]])
```

In [32]:
```
labels
```

Out[32]:
```
array([4, 0, 0, ..., 5, 1, 2])
```

In [33]:
```
from sklearn import metrics
silhouette_score = metrics.silhouette_score(X_train, labels, metric='euclidean')
```

In [34]:
```
silhouette_score
```

Out[34]:
```
0.16370979
```

In [35]:
```
import numpy as np
```

In [36]:
```
data_point=cluster_center[1]
```

In [37]:
```
centroid=cluster_center[2]
```

In [38]:
```
distance = np.sqrt(np.sum((data_point - centroid)**2))
```

In [39]:
```
distance
```

Out[39]:
```
3.115856770026144
```

In [40]:
```
data_point=cluster_center[2]
centroid=X_train[4]
distance = np.sqrt(np.sum((data_point - centroid)**2))
distance
```

Out[40]:
```
3.555651305596432
```

```
In [41]: 1 cluster_center[1]
```

```
Out[41]: array([ 0.45112201,  0.27994382,  0.81651045,  0.18406322,  0.66813233,
                -0.04907938,  0.45178895,  0.67359511,  0.23057321,  0.36369726,
                -0.21904267,  0.57521646,  0.48674021, -0.67717443, -0.24077094,
                 0.10005647, -0.75129657,  0.13301864,  0.04524492,  0.63950955,
                -0.46642303,  0.33890124, -0.10893337, -0.61050981, -1.01739013,
                 0.18615144,  0.20548162, -0.15630597,  0.44121739,  0.24071426,
                 0.05122652, -0.46033426,  0.12162386, -0.25628653, -0.03556913,
                 0.23338878, -0.81635451,  0.44172316, -0.80352692, -0.12454932,
                -0.07965993,  0.17218068,  0.832853  , -0.48514729,  0.58639443,
                 0.52545852,  0.01606144,  0.08033636, -1.10127285, -0.57735659,
                 0.89576774, -0.18772205,  0.08368704, -0.3536889 ,  0.23673471,
                -0.23860111,  0.39517127,  0.41651229,  0.33984588, -0.2609618 ,
                 0.3774438 , -1.22137639, -0.06257518, -0.37866958, -0.08421788,
                -0.22165313, -0.55139268, -0.46526928, -0.20305721, -0.18328147,
                 0.57743268, -0.22588137, -0.15999532, -0.09880484, -0.21025995,
                -0.11121118, -1.06081152,  0.23002972,  0.24577975, -0.34248069,
                -0.66952963, -0.52698797, -0.08437221,  0.2006609 ,  0.41812382,
                -0.64018055, -0.36857677,  0.09587825,  0.32604233,  0.02189478,
                 0.56991184, -0.14581243, -0.17592221,  0.30455307,  0.11514485,
```

```
In [42]: 1 X_train[0]
```

```
Out[42]: array([ 0.43752038,  0.2750144 ,  0.66334444,  0.04836331,  0.5210062 ,
                -0.15144932,  0.26334286,  0.6657086 ,  0.34489164,  0.4130081 ,
                -0.36253873,  0.50701106,  0.62280685, -0.473016  , -0.19060749,
                 0.06559386, -0.6046061 ,  0.11846886,  0.05783413,  0.6227517 ,
                -0.4819998 ,  0.3758282 , -0.11798802, -0.49917546, -0.7975715 ,
                 0.21456014,  0.23157969, -0.07828102,  0.40322873,  0.2571515 ,
                -0.02172235, -0.43344066,  0.06130793, -0.18308659, -0.02023095,
                 0.13678853, -0.7670587 ,  0.4113384 , -0.6616126 , -0.07470948,
                -0.04868703,  0.3088321 ,  0.57074773, -0.29375616,  0.4470927 ,
```

```
In [43]: 1 x1=list(X_train)
```

```
In [44]: 1 plot_list1=[]
```

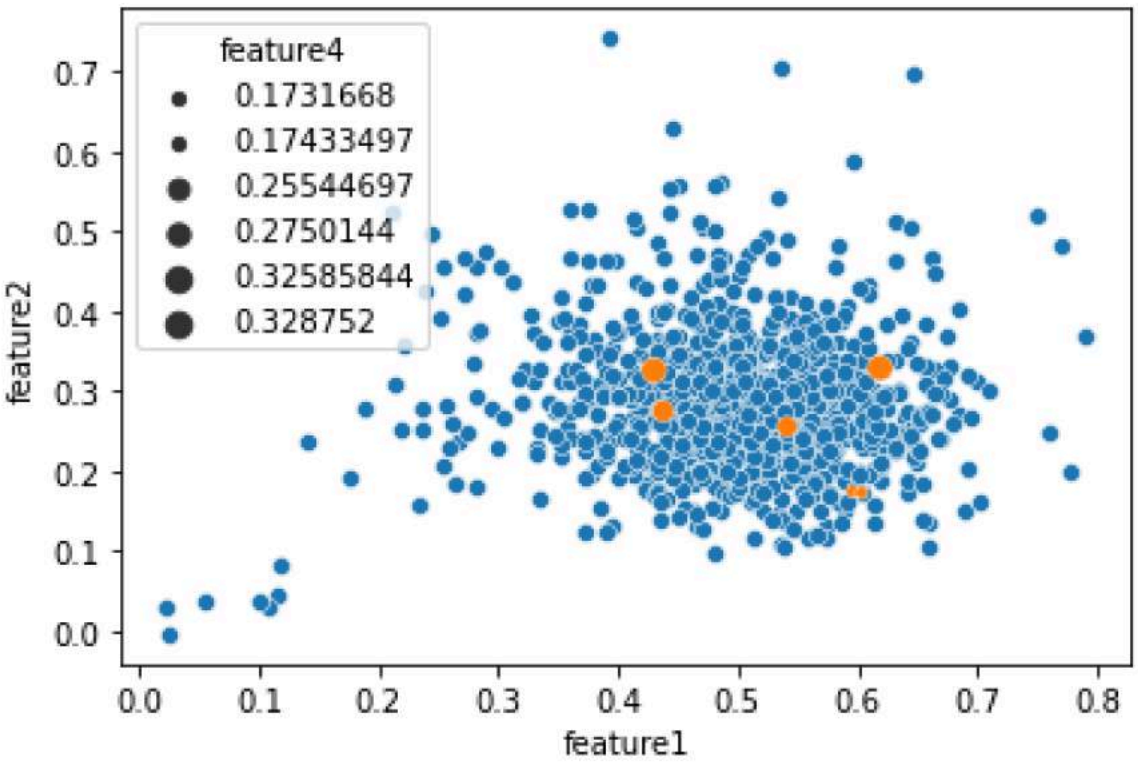
```
In [45]: 1 cluster_center
```

```
Out[45]: array([[ 0.51205407,  0.26649424,  0.90241309, ..., -0.09212171,
                  -0.37043781,  0.57423351],
                [ 0.45112201,  0.27994382,  0.81651045, ..., -0.1418154 ,
                  -0.36768555,  0.55361764],
                [ 0.4415233 ,  0.41966983,  0.88872495, ..., -0.38047016,
                  -0.680373  ,  0.64575728],
                [ 0.06373378,  0.04069745,  0.14572571, ..., -0.07735449,
                  -0.11231906,  0.10867206],
                [ 0.3268094 ,  0.2722219 ,  0.69805395, ..., -0.27002691,
                  -0.41384042,  0.50126829],
                [ 0.54242459,  0.27594353,  0.96211601, ..., -0.06060533,
                  -0.42206391,  0.58879165]])
```

```
In [46]:
1 plot_list3=[]
2 for pp in range(len(cluster_center)):
3
4     plot_list3.append(x1[pp][0])
5
6
7 plot_list4=[]
8 for pp in range(len(cluster_center)):
9
10    plot_list4.append(x1[pp][1])
11
```

In [49]:
```
# inputs
feature1 = plot_list1[0:1000]
feature2= plot_list2[0:1000]
feature3= plot_list3
feature4= plot_list4
# convert to pandas dataframe
d = {'feature1': feature1, 'feature2': feature2}
pdnumsqr = pd.DataFrame(d)


d1 = {'feature3': feature3,'feature4': feature4}
pdnumsqr1 = pd.DataFrame(d1)
# plot using lineplot
#sns.set(style='darkgrid')
#sns.lineplot(x='num', y='sqr', data=pdnumsqr)

sns.scatterplot(x='feature1', y='feature2', data=pdnumsqr)
sns.scatterplot(x='feature3', y='feature4', data=pdnumsqr1, size="feature4")
```

Out[49]: <AxesSubplot:xlabel='feature1', ylabel='feature2'>

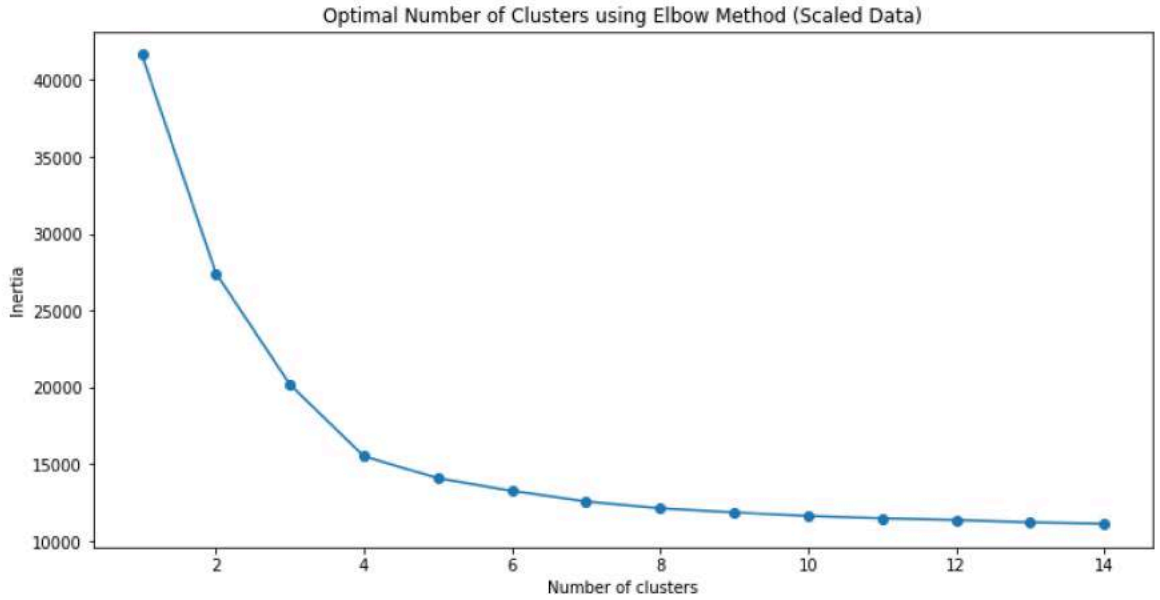


We have to find the optimal K value for clustering the data. Now we are using the Elbow method to find the optimal K value.

In [50]:
```
sse = []
k_list = range(1, 15)
for k in k_list:
    km = KMeans(n_clusters=k)
    km.fit(X_train)
    sse.append([k, km.inertia_])

oca_results_scale = pd.DataFrame({'Cluster': range(1,15), 'SSE': sse})
plt.figure(figsize=(12,6))
plt.plot(pd.DataFrame(sse)[0], pd.DataFrame(sse)[1], marker='o')
plt.title('Optimal Number of Clusters using Elbow Method (Scaled Data)')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

Out[50]:
```
Text(0, 0.5, 'Inertia')
```

The point at which the elbow shape is created is 7, that is, our K value or an optimal number of clusters is 7. Now let's train the model on the dataset with a number of clusters 7.



```
df_scale2 = X_train
kmeans_scale = KMeans(n_clusters=6, n_init=100, max_iter=400, init='k-means++', random_state=42).fit(df_scale2)
#print('KMeans Scaled Silhouette Score: {}'.format(silhouette_score(df_scale2, kmeans_scale.labels_, metric='euclidean')))
labels_scale = kmeans_scale.labels_
clusters_scale = pd.concat([df, pd.DataFrame({'cluster_scaled':labels_scale})], axis=1)
```

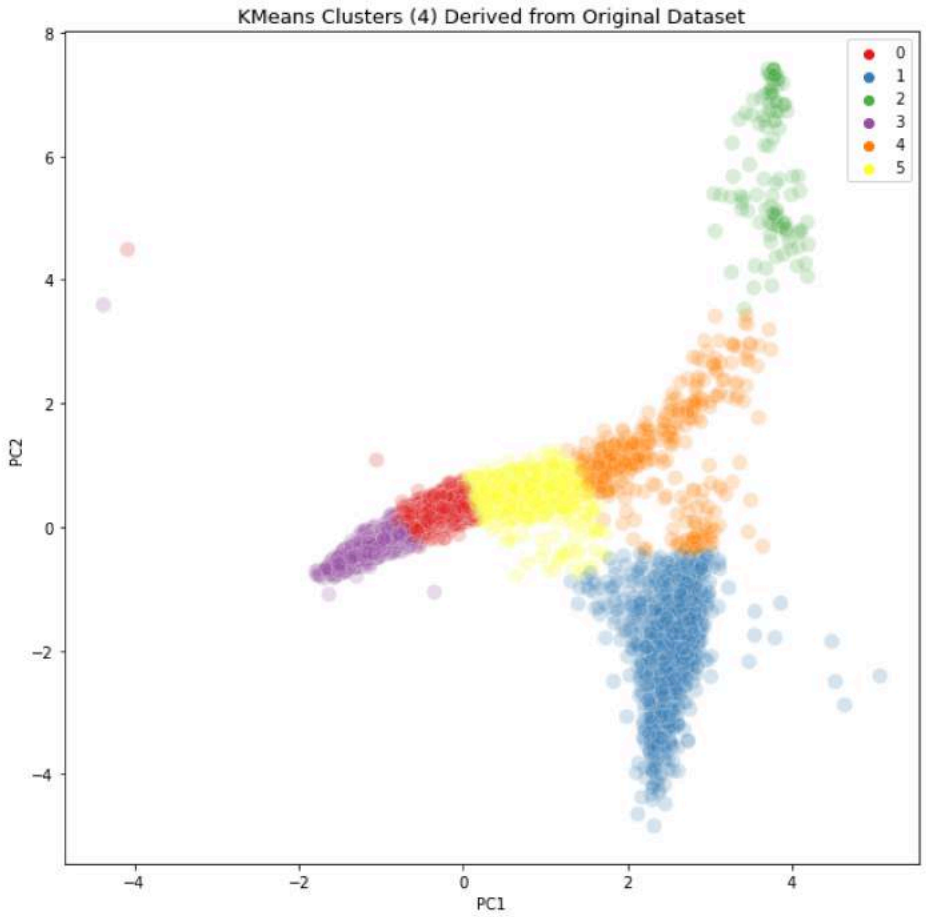
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.metrics import silhouette_score
```

```
pca2 = PCA(n_components=3).fit(X_train)
pca2d = pca2.transform(X_train)
plt.figure(figsize = (10,10))
sns.scatterplot(pca2d[:,0], pca2d[:,1],
                hue=labels_scale,
                palette='Set1',
                s=100, alpha=0.2).set_title('KMeans Clusters (4) Derived from Original Dataset', fontsize=13)
plt.legend()
plt.ylabel('PC2')
plt.xlabel('PC1')
plt.show()
```

In the above piece of code blocks, The StandardScaler() method in the Python Sklearn package allows us to normalize the data values into a common format.

We initially build an object of the StandardScaler() method in accordance with the aforementioned syntax. Additionally, we normalize the data by using fit transform() together with the provided object.

In conclusion in our analysis of the clusters, I can see that they make logic, and I would definitely consider these to be a solid balance of musical styles and lyrical content. We frequently have user data and other elements in production-type recommendation systems that can make recommendations more sophisticated and accurate. With the resources we had, we did our best.



kmeans give us different clusters correspondingly. Now let's plot all the clusters using matplotlib.As you can see there are clusters in total which are visualized in different colors and the centroid of each cluster is visualized in black color.

Chapter 4

Code for building clustered documents using word2vec using Kmeans and Discussion with Results

4.1 Introduction

we will learn how to cluster documents using Word2Vec, in this Chapter, we'll train a Word2Vec model, generate word embeddings, and use K-means to create groups of songs using their lyrics and genres. some parts of the code blocks used in the previous chapter will also be used in this chapter to know the complete flow of implementing the project. and we will see what packages we used to build the word2vec model.

More people work on sentiment analysis with the supervised learning method which receives more attention than other NLP models. In order to gain a deeper understanding of Natural Language Processing, we must use unsupervised learning methods for a majority of data (text). You'll discover how to use unsupervised learning in this thesis project to extract value from your text data. By using the K-means method with word embedding training (Word2Vec), you may cluster texts.

4.2 How to cluster document?

These are usually 3 Steps carried out during Clustering Process.

1. Lowercasing text, eliminating non-alphanumeric letters, or stemming words are typical methods for cleaning and tokenizing data.

2. The process of creating vector representations of the documents entails translating them from words to numerical vectors; two popular techniques for achieving this are word embeddings and bag-of-words models.

3. It is necessary to choose and use a clustering technique in order to apply it to the document vectors in order to discover the best groups feasible. Algorithms like K-means, DBSCAN, and hierarchical clustering are often utilized.

4.2.1 Set Up Your Environment

First and Foremost, we need to set our local machine with the following libraries and tools.

- Install anaconda in Your local system and we can have a complete set of environments.
- create a new environment with conda or venv.
- Activate Your new environment and launch the jupyter notebook.

Here is the piece of code for these above steps.

- conda create –name venv
- conda activate venv
- pip install -r requirements
- jupyter notebook

Next, open Jupyter Notebook. Then, create a new notebook(python file) in the root folder and set its name to "final developemnt.ipynb".From here you will see my own code blocks.

4.2.2 The Software packages used for Word2vec model

This chapter is dedicated to the complete description of the codes used in building the Doc2vec model. The packages that have been used in this work have been explained in previous chapters but here we will just see some of the libraries that are used. The Python code is written in a local environment as I mentioned in the above section,as I have installed anaconda which they have built-in python installed as a prerequisite.

Loading Libraries

In [1]:
```
import pandas as pd
import numpy as np
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")

from sklearn import utils
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

from gensim.models import Doc2Vec
import gensim
from gensim.models.doc2vec import TaggedDocument

import re
import seaborn as sns
import matplotlib.pyplot as plt



# Erase Warnings
import warnings
warnings.filterwarnings("ignore")
```

This code imports all the necessary libraries like pandas, numpy, and sklearn libraries like train and test split models from sklearn.

You can see all of the columns of the dataset in the output above. It is able to find similarities to music due to all its audio features.

Before moving forward, I will not use some of its column in Dataframe as it is of no use like Playlist id,track id and many more as you can see complete DataFrame in the below figure.

These are listed columns that we wont use much in our work. track id, track name, track artist, track popularity, track album id, track album name, track album release date, playlist name, playlist id

```
In [6]: songs_df.shape, songs_df.columns, songs_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18454 entries, 0 to 18453
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   track_id                 18454 non-null  object
 1   track_name               18454 non-null  object
 2   track_artist             18454 non-null  object
 3   lyrics                   18194 non-null  object
 4   track_popularity         18454 non-null  int64
 5   track_album_id           18454 non-null  object
 6   track_album_name         18454 non-null  object
 7   track_album_release_date 18454 non-null  object
 8   playlist_name            18454 non-null  object
 9   playlist_id              18454 non-null  object
 10  playlist_genre           18454 non-null  object
 11  playlist_subgenre        18454 non-null  object
 12  danceability             18454 non-null  float64
 13  energy                   18454 non-null  float64
 14  key                      18454 non-null  int64
 15  loudness                 18454 non-null  float64
 16  mode                     18454 non-null  int64
 17  speechiness              18454 non-null  float64
 18  acousticness             18454 non-null  float64
 19  instrumentalness         18454 non-null  float64
 20  liveness                 18454 non-null  float64
 21  valence                  18454 non-null  float64
 22  tempo                    18454 non-null  float64
 23  duration_ms              18454 non-null  int64
 24  language                 18194 non-null  object
dtypes: float64(9), int64(4), object(12)
memory usage: 3.5+ MB
Out[6]: ((18454, 25),
 Index(['track_id', 'track_name', 'track_artist', 'lyrics', 'track_popularity',
        'track_album_id', 'track_album_name', 'track_album_release_date',
        'playlist_name', 'playlist_id', 'playlist_genre', 'playlist_subgenre',
        'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
        'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
        'duration_ms', 'language'],
       dtype='object'),
 None)
```

As you can see in this code block, the Dataset Details are shown, such as the names of the columns and their data types.

By doing this, we can find out how much data is stored in the dataset and what information it contains. Whether the data type is an object, float or integer type, we can easily find out what it is.

	track_popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence
track_popularity	1.000000	0.061499	-0.093521	-0.006878	0.031290	0.010947	-0.000622	0.076821	-0.080933	-0.059334	-0.004490
danceability	0.061499	1.000000	-0.069708	0.002090	0.022830	-0.078020	0.204041	-0.029240	-0.049187	-0.114719	0.343419
energy	-0.093521	-0.069708	1.000000	0.018833	0.674795	-0.009903	-0.011986	-0.546033	0.042093	0.156931	0.207941
key	-0.006878	0.002090	0.018833	1.000000	0.007280	-0.169589	0.026282	-0.002067	0.004243	0.000682	0.021927
loudness	0.031290	0.022830	0.674795	0.007280	1.000000	-0.035276	0.029658	-0.370628	-0.099164	0.071159	0.050211
mode	0.010947	-0.078020	-0.009903	-0.169589	-0.035276	1.000000	-0.070013	0.022191	-0.002906	0.004850	-0.009382
speechiness	-0.000622	0.204041	-0.011986	0.026282	0.029658	-0.070013	1.000000	-0.000217	-0.108365	0.060072	0.035202
acousticness	0.076821	-0.029240	-0.546033	-0.002067	-0.370628	0.022191	-0.000217	1.000000	-0.015803	-0.067491	-0.069817
instrumentalness	-0.080933	-0.049187	0.042093	0.004243	-0.099164	-0.002906	-0.108365	-0.015803	1.000000	-0.000021	-0.103020
liveness	-0.059334	-0.114719	0.156931	0.000682	0.071159	0.004850	0.060072	-0.067491	-0.000021	1.000000	-0.006931
valence	-0.004490	0.343419	0.207941	0.021927	0.050211	-0.009382	0.035202	-0.069817	-0.103020	-0.006931	1.000000
tempo	0.011954	-0.201465	0.138291	-0.018967	0.082558	0.026421	0.033620	-0.096149	0.028126	0.015658	-0.017101
duration_ms	-0.143970	-0.133603	-0.021134	0.016984	-0.159026	0.012433	-0.094343	-0.052996	0.078505	0.026988	-0.045546

In addition, we will learn about Spotify's features like song popularity, loudness, and others.

The code below illustrates how we can use doc2vec embedding techniques and the DBOW model to further work on our main word embedding model by combining song genres, lyrics, and languages.

On basis of the model is only based on these three columns, we combined all three columns and created a new column.

Combining genres + lyrics + labguages and creating a fetured document

In [9]:
```
df = pd.read_csv('final_spotify_songs.csv')
df = df[['playlist_genre','lyrics','language']]

df.head(10)
```

Out[9]:

	playlist_genre	lyrics	language
0	rock	Minsan pa Nang ako'y napalingon Hindi ko alam ...	tl
1	rock	The trees, are singing in the wind The sky blu...	en
2	r&b	NA Yeah, Spyderman and Freeze in full effect U...	en
3	r&b	I really can't stay Baby it's cold outside I'v...	en
4	pop	Get up out of my business You don't keep me fr...	en
5	r&b	Hold your breath, don't look down, keep trying...	en
6	r&b	All I want is somebody who's gonna love me for...	en
7	r&b	Feels good Everybody Tender lover Tender love ...	en
8	edm	Don't run away, it's getting colder Our hearts...	en
9	r&b	Ho una cosa da dirti da tempo Ma non ho mai t...	it

As part of this code block, I am printing the first output of combined columns as one document. Usually, this process takes a short while, but can take a long time depending on the system's requirements. Our document as a whole looked like this after we combined it in the next line of code. as you can see the result in below snapshot of code.

In [19]:
```
def print_complaint(index):
    example = df[df.index == index][['lyrics', 'playlist_genre']].values[0]
    if len(example) > 0:
        print(example[0])
        print('playlist_genre:', example[1])

print_complaint(2)
```

```
r&b-NA Yeah, Spyderman and Freeze in full effect Uh-huh You ready, Ron? I'm ready You ready, Biv? I'm ready, Slick, are you? Oh, yeah, break it down N
A Girl, I, must (warn you) I sense something strange in my mind Situation is (serious) Let's cure it cause we're running out of time It's oh, so (beau
tiful) Relationships they seem from the start It's all so (deadly) When love is not together from the heart It's drivin' me out of my mind! That's why
it's HARD for me to find Can't get it out of my head! Miss her, kiss her, love her(Wrong move you're dead!) That girl is (poison)...Never trust a big
butt and smile That girl is (poison)..("POISON!!") NA (-caution) Before I start to meet a fly girl, you know? Cause in some (portions) You'll think sh
e's the best thing in the world She's so - (fly) She'll drive you right out of your mind And steal your heart when you're blind Beware she's schemin',
she'll make you think you're dreamin' YOU'LL fall in love and you'll be screamin', demon, HOO.. Poison, deadly, movin' in slow Lookin for a mellow fel
low like DeVoe Gettin paid, laid, so better lay low Schemin on house, money, and the whole show The low pro ho she'll be cut like an aaa-FRO See what
you're sayin', huh, she's a winner to you But I know she's a loser (How do you know?) Me and the crew used to do her! "POISON!" "POISON!" "POISON!" "P
OISON!" "POISON!" "POISON!" "POISON!" "POISON!" "POISON!" "POISON!" "POISON!" "POISON!" "POISON!" "POISON!" "POISON! I was at the park, shake,
breakin and takin 'em all And that night, I played the wall Checkin' out the fellas, the highs and lows Keepin' one eye open, still clockin' the hoes
There was one particular girl that stood out from the rest Poison as can be, the high power chest Michael Biv here and I'm runnin' the show Bell, Biv
DeVoe ..now you know! Yo, Slick, blow.. It's drivin' me out of my mind! That's why it's HARD for me to find Can't get it out of my head! Miss her, kis
s her, love her(Wrong move you're dead!) That girl is (poison)...Never trust a big butt and smile That girl is (poison)..("POISON!!") Yo' fellas, that
was my end of.. You know what I'm sayin', Mike? Yeah, B.B.D. in full effect Yo', wassup to Ralph T and Johnny G And I can't forget about my boy, B. Br
own And the whole NE crew Poison.. NA-en
playlist_genre: r&b
```

Using the beautiful soup library, we can delete any links of any type we detect in the data. Therefore, Beautiful Soup is essentially a Python module for parsing HTML and XML texts (including having malformed markup, i.e. non-closed tags, so named after tag soup). For processed pages, it generates a parse tree that may be used to extract HTML data for web scraping. However, in our instance, we are only utilizing this package to eliminate any unwanted hyperlinks that may have got into the dataset.

Having combined the text, we can now move on to cleaning it up, so we will be performing a cleaning process in the document and tokenizing the data. The data you will use for clustering needs to be read and preprocessed after you import the libraries. The preprocessing consists of cleaning and tokenizing the data.

Removing stopwords (eliminate unimportant words, allowing applications to focus on the important words instead)

In [21]:
```
train, test = train_test_split(df, test_size=0.3, random_state=42)
import nltk
from nltk.corpus import stopwords
def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            if len(word) < 2:
                continue
            tokens.append(word.lower())
    return tokens

train_tagged = train.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['lyrics']), tags=[r.playlist_genre]), axis=1)
test_tagged = test.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['lyrics']), tags=[r.playlist_genre]), axis=1)
```

In [22]:
```
train_tagged.values[2]
```

Out[22]:
```
TaggedDocument(words=['pop-i', 'woke', 'up', 'it', 'was', 'waited', "'til", '11', 'just', 'to', 'figure', 'out', 'that', 'no', 'one', 'would', 'call',
'think', "'ve", 'got', 'lot', 'of', 'friends', 'but', 'do', "n't", 'hear', 'from', 'them', 'what', "'s", 'another', 'night', 'all', 'alone', 'when',
'you', "'re", 'spending', 'everyday', 'on', 'your', 'own', 'and', 'here', 'it', 'goes', "'m", 'just', 'kid', 'and', 'life', 'is', 'nightmare', "'m",
'just', 'kid', 'know', 'that', 'it', "'s", 'not', 'fair', 'nobody', 'cares', "'cause", "'m", 'alone', 'and', 'the', 'world', 'is', 'having', 'more',
'fun', 'than', 'me', 'tonight', 'and', 'maybe', 'when', 'the', 'night', 'is', 'dead', "'ll", 'crawl', 'into', 'my', 'bed', "'m", 'staring', 'at', 'the
se', 'four', 'walls', 'again', "'ll", 'try', 'to', 'think', 'about', 'the', 'last', 'time', 'had', 'good', 'time', 'everyone', "'s", 'got', 'somewher
e', 'to', 'go', 'and', 'they', "'re", 'gon', 'na', 'leave', 'me', 'here', 'on', 'my', 'own', 'and', 'here', 'it', 'goes', "'m", 'just', 'kid', 'and',
'life', 'is', 'nightmare', "'m", 'just', 'kid', 'know', 'that', 'it', "'s", 'not', 'fair', 'nobody', 'cares', "'cause", "'m", 'alone', 'and', 'the',
'world', 'is', 'having', 'more', 'fun', 'than', 'me', 'what', 'the', 'hell', 'is', 'wrong', 'with', 'me', 'do', "n't", 'fit', 'in', 'with', 'anybody',
'how', 'did', 'this', 'happen', 'to', 'me', 'wide', 'awake', "'m", 'bored', 'and', 'ca', "n't", 'fall', 'asleep', 'and', 'every', 'night', 'is', 'th
e', 'worst', 'night', 'ever', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid', 'yeah', "'m", 'just', 'kid', "'m",
'just', 'kid', "'m", 'just', 'kid', 'and', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid', "'m", 'just', 'kid', 'and', 'life', 'is', 'n
ightmare', "'m", 'just', 'kid', 'know', 'that', 'it', "'s", 'not', 'fair', 'nobody', 'cares', "'cause", "'m", 'alone', 'and', 'the', 'world', 'is', 'n
obody', 'wants', 'to', 'be', 'alone', 'in', 'the', 'world', "'m", 'just', 'kid', 'and', 'life', 'is', 'nightmare', "'m", 'just', 'kid', 'know', 'tha
t', 'it', "'s", 'not', 'fair', 'nobody', 'cares', "'cause", "'m", 'alone', 'and', 'the', 'world', 'is', 'nobody', 'wants', 'to', 'be', 'alone', 'in',
'the', 'world', 'nobody', 'cares', "'cause", "'m", 'alone', 'and', 'the', 'world', 'is', 'having', 'more', 'fun', 'than', 'me', 'tonight', "'m", 'al
l', 'alone', 'tonight', 'nobody', 'cares', 'tonight', "'cause", "'m", 'just', 'kid', 'tonight-en'], tags=['pop'])
```

In this code block, We apply the stopword library, a group of frequently used terms in any language, Stop words in English include "the," "is," and "and," for instance. Stop words are used in NLP and text mining applications to filter out unnecessary words so that the keywords may be the focus.

As you can see from the code, we are using the standard data science modeling method of partitioning the data into train and test data. The remaining 20% of the data is utilized

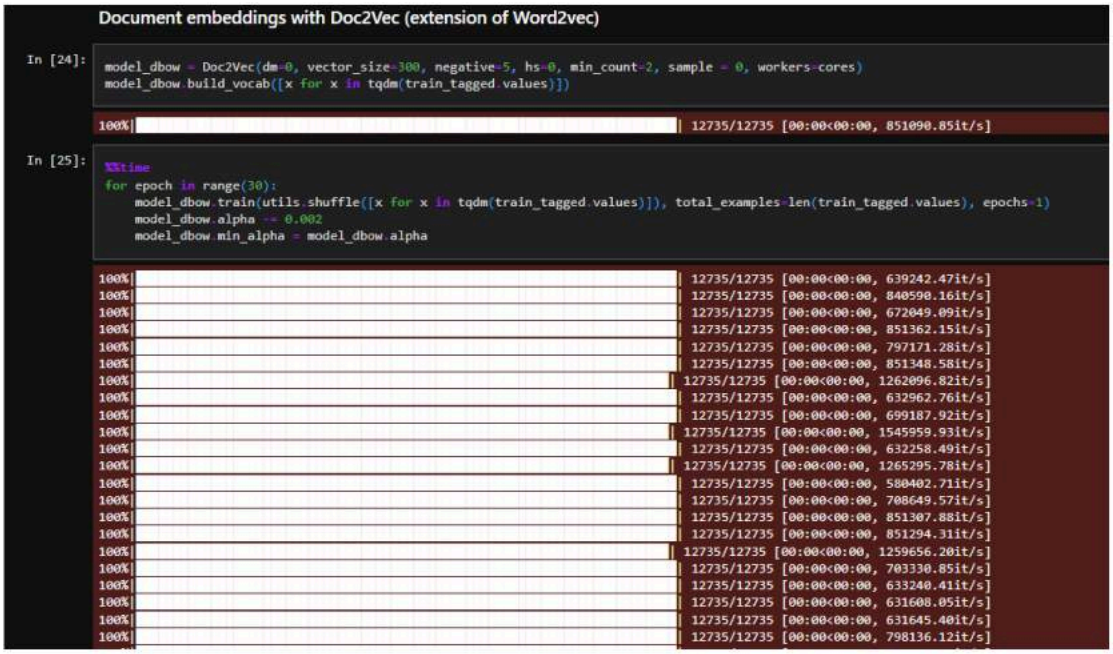
for testing, with the remaining 80% being used for training. As a result, I am utilizing the classes train and test in the first line, and we are assigning the test size of 0.3 and the random state of 42 using the train test split function.

let's understand why we use a Random state.

To make sure that the outcomes we acquire can be replicated, we use a Random state to establish the seed for the random generator. You would obtain different data allocated to the train and test data unless you could adjust for the random factor since the nature of separating the data into train and test is that it is randomized.

The trained tagged values are just the trained data in a tokenized format, and in the following piece of code, we can see them. Each phrase in the lyrical document is tokenized into a single word.

Now we can see how the dbow model functions for document embeddings in the following steps.



Doc2Vec Model Training

You may train a Doc2Vec model with the aid of the following code.

A dbow model is setup using the first line of code. The document we prepared before using name train tagged values is utilized to train this model in the second line of code.

Using your tokenized documents as a basis, you use this code to train a Doc2Vec model. You set the following arguments in the Doc2Vec class for this example:

- The tokenized documents should be in a list of lists, as expected by trained tagged values.
- The word vectors' size is defined by vector size. In this instance, I have chosen 300.
- All words with a total frequency lower than this are ignored by the min count (int, optional) function.
- Negative sampling - negative (optional integer) The int for negative defines how many "noise words" should be drawn if negative sampling is utilized and > 0, (usually between 5-20). If set to 0, there is no negative sampling. in this instance, I have used 5.
- How many cores are used for training depends on the workers. To ensure that the code is deterministically replicable, I set workers as cores.
- You can adjust additional factors when developing the Doc2Vec model. For additional information about gensim, go to its documentation.
- We will generate the document model after the parameter has been specified in accordance with your requirements.
- Its common values vary from 100 (the gensim default, for speed and memory-compactness) to 1000, depending on the size in dimensions of the word-vectors/doc-vectors that are formed. For word vectors, values between 300 and 400 seem to be particularly typical.

As we can see, we have produced a distributed bag of words version of the Doc2Vec model called DBOW. By estimating the vector for a new document, let's see how it is handling document embedding.

```
def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors

def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors
```

```
y_train, X_train = vec_for_learning(model_dbow, train_tagged)
y_test, X_test = vec_for_learning(model_dbow, test_tagged)
```

```
X_train[0]
```

```
array([ 0.12776385, -0.34752703,  0.5917772 , -0.4151879 ,  0.24446405,
        0.19634247, -0.17245373, -0.3145219 , -0.16556837,  0.8384734 ,
        0.24948435,  0.19812529, -0.09236781, -0.7852816 , -0.6394208 ,
        0.53060925, -0.41110063,  0.37662256, -0.20300746, -0.17487586,
       -0.05667151, -0.28930315, -0.381548  , -0.736375  , -0.26568067,
       -0.36380497,  0.30748245,  0.08442096,  0.22642122,  0.44361725,
        0.05440766, -0.14296634, -0.1761249 , -0.56881636,  0.18493986,
       -0.15294679,  0.17976733,  0.6629474 , -0.20629647,  0.40015942,
       -0.05089117, -0.24609964,  0.35218936, -0.4626624 ,  0.5260573 ,
        0.08539285,  0.58064514, -0.40153688, -0.38637626, -1.082219  ,
        0.6153424 , -0.42065603,  0.5509572 , -0.5536649 ,  0.16550022,
        0.79277766,  0.6752136 , -0.36678347,  0.6510203 ,  0.32869446,
        0.04762083, -0.4560007 ,  0.29112023,  0.17744328, -0.26977062,
       -0.323952  ,  0.05611678, -0.00627052,  0.37853196, -0.7804238 ,
        0.06897839,  0.2566141 ,  0.6243154 ,  0.42753303, -0.24461223,
       -0.13439322, -0.6036048 ,  0.44090733, -0.26068696, -0.4293626 ,
        0.5250061 , -0.02198012,  0.4805842 , -0.468162  ,  0.08439224,
       -0.655567  ,  0.0774563 , -0.09561424,  0.4041368 ,  0.40093765,
        0.07745641,  0.1572914 ,  0.07142451, -0.33399227,  0.0474212 ,
```

Cosine similarity can be used to compare the above output with other vectors. The model we trained is built on approximation iterative techniques, therefore there is a chance that subsequent inferences of the same text will result in different vectors. The phrases we used with the infer vector module are also shown in the output from the previous line as vector representations.

so we have stored all our numerical format data in X train and Y train objects and now we will work on those objects to analyze them for the clustering using K-means since I don't explain much about K-means in this section. we will probably see the working method.
AS I have explained in detail about the implementation of kmeans clustering analysis in chapter 3 of (section 3.7) and same method is applied to the genre and sub genre analysis.

4.2.3 TSNE Feature Vectorization over the Genres and Languages

Now that we've reached this area, let's move on to the feature vectors, where we may use the Spotify-defined features for songs Data using TSNE.

TSNE is mostly utilized to understand high-dimensional data and project it into low-dimensional space (like 2D or 3D). That makes it very helpful when working with models of neural networks.

Therefore, points in the high-dimensional data set that are near to one another will typically be close to one another on the graph. Additionally, t-SNE generates stunning visuals.

Let's code for the implementation of t-SNE over the genres and languages using Spotify features.

```
features = ["track_popularity", "danceability", "energy", "key",
            "loudness", "mode", "speechiness" ,"acousticness",
            "instrumentalness", "liveness", "valence", "tempo", "duration_ms"]

feature_vector = songs_df[features]
```

```
tsne = TSNE(n_components=2, verbose=1, random_state=0)
w_features = tsne.fit_transform(feature_vector)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 18454 samples in 0.231s...
[t-SNE] Computed neighbors for 18454 samples in 0.736s...
[t-SNE] Computed conditional probabilities for sample 1000 / 18454
[t-SNE] Computed conditional probabilities for sample 2000 / 18454
[t-SNE] Computed conditional probabilities for sample 3000 / 18454
[t-SNE] Computed conditional probabilities for sample 4000 / 18454
[t-SNE] Computed conditional probabilities for sample 5000 / 18454
[t-SNE] Computed conditional probabilities for sample 6000 / 18454
[t-SNE] Computed conditional probabilities for sample 7000 / 18454
[t-SNE] Computed conditional probabilities for sample 8000 / 18454
[t-SNE] Computed conditional probabilities for sample 9000 / 18454
[t-SNE] Computed conditional probabilities for sample 10000 / 18454
[t-SNE] Computed conditional probabilities for sample 11000 / 18454
[t-SNE] Computed conditional probabilities for sample 12000 / 18454
[t-SNE] Computed conditional probabilities for sample 13000 / 18454
[t-SNE] Computed conditional probabilities for sample 14000 / 18454
[t-SNE] Computed conditional probabilities for sample 15000 / 18454
[t-SNE] Computed conditional probabilities for sample 16000 / 18454
[t-SNE] Computed conditional probabilities for sample 17000 / 18454
[t-SNE] Computed conditional probabilities for sample 18000 / 18454
[t-SNE] Computed conditional probabilities for sample 18454 / 18454
[t-SNE] Mean sigma: 88.874677
[t-SNE] KL divergence after 250 iterations with early exaggeration: 69.083008
[t-SNE] KL divergence after 1000 iterations: 0.568073
```

The features from Spotify are used in the code above, and they are transformed using the tsne fit transform technique on these feature vectors. The nearest neighbors are calculated using an iterative approach with 1000 iterations for each of the indexed variables.

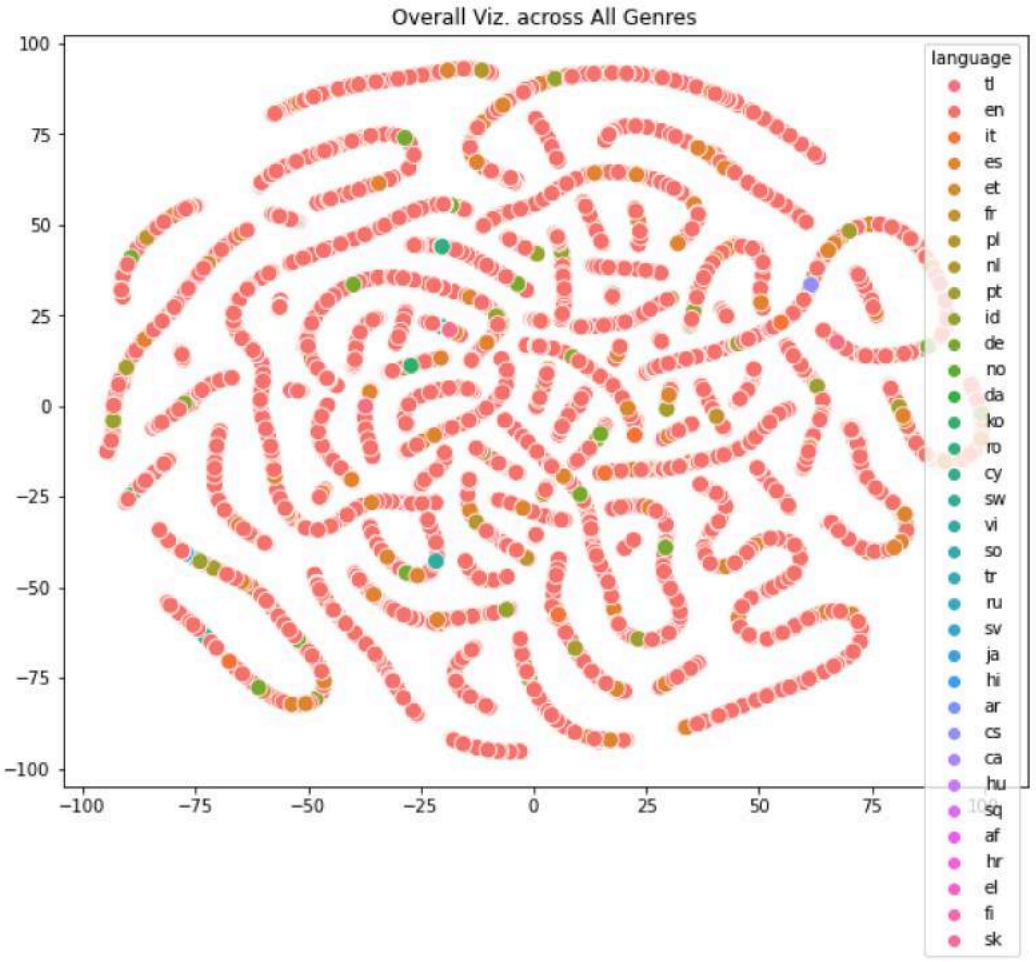
After obtaining that matrix for each song, I used t-SNE to build a 2D embedding. In the end, I just created the 2D chart with the genres and languages on it as the map. For that purpose, it is simple to tell which songs are from various languages because they are "similar" to one another. The data and attributes produced by Spotify may explain why the graph has a snaky appearance.

```
len(w_features)
```

```
18454
```

```
comp0 = w_features[:,0]
comp1 = w_features[:,1]
sns.scatterplot(x=comp0, y=comp1,s=100, hue=songs_df["language"],
                data=feature_vector).set(title="Overall Viz. across All Genres"),
# Change seaborn plot size
fig = plt.gcf()
fig.set_size_inches(10, 8)
```

In the below Figure You can see the tsne visualised graph for all the genres based on Langugaes.



4.3 Distance between Genres with Final Results

The purpose of this chapter is to explain the results and discuss the DBOW model that we used for genres and subgenres based on languages and lyrics in order to document and work on it. In our result section, we can see results separately. I will try to explain everything as much as possible. We have included the summary of the distance between genres and subgenres with dbow model with the objective of this thesis work in this section in order to have a holistic representation of our work.

4.3.1 How we found out the distance?

In this section, we will compare the distance between clusters and their centroid points mathematically and visually and also its implementation.

The main objective of this thesis is cluster analysis and finding out the distances between cluster centroid points of each cluster.

Here cluster represents the group of genres. In the main genre, we can see 6 clusters which means 6 genres which I mentioned in the above section. as You will see the clustered points(genres and subgenres) that are grouped according to their similarity can be seen in the above picture in section K-means.

In order to describe genres and subgenres with labels and cluster centers after establishing clustered groupings of them, let's first define labels. Labels are just the train model that we saw in the preceding line of code when applying the dbow model to the genre analysis.

In the below line of code, you can see the distance between genres and its results with code.

So we will use Numpy to calculate the distance between genres. The Euclidean distance is the distance between two points that is shortest, regardless of their dimensions. In this section, we will use the Numpy library to find the Euclidean distance. This library is used to

manipulate multidimensional arrays very efficiently, so let's discuss and see code to figure out the Euclidean distance with the Numpy library.

Since we imported all the libraries needed, we will be able to view the code for the implication directly in our thesis model. Here is the first piece of code.

```
#applying on given lyrics and sub gener and plot the centroides and distances
```

```
labels = kmeans_scale.labels_
cluster_center=kmeans_scale.cluster_centers_
```

```
labels
```

```
array([5, 3, 3, ..., 2, 0, 3])
```

```
cluster_center
```

```
array([[-7.77378049e-02, -3.38392734e-02,  2.06328021e-01, ...,
         9.65537120e-01,  4.33135360e-01, -2.08374137e-01],
       [ 5.76515123e-03, -1.16242200e-02,  4.85565235e-02, ...,
         2.37686093e-01,  9.67731908e-02, -1.45587451e-02],
       [-1.61821786e-01, -2.29255623e-02,  2.10903904e-01, ...,
         1.10816231e+00,  4.89212991e-01, -3.03132384e-01],
       [-1.33634653e-01, -3.00720755e-02,  2.13753892e-01, ...,
         1.03893705e+00,  4.53312424e-01, -2.52252392e-01],
       [-3.91146036e-02,  5.59631711e-03,  3.85950893e-04, ...,
         9.71507105e-01,  3.95842439e-01, -3.09369468e-01],
       [-5.77229725e-03, -2.68687762e-02,  1.78810985e-01, ...,
         8.59953145e-01,  4.18854262e-01, -1.48901027e-01]])
```

After importing the necessary libraries we directly see the labels and cluster centers values as you can see in the above code.

A label is simply a vector representation of a trained model for genres and lyrics which are considered to be scaled labels over the data. Cluster centers are nothing more than centroid points of the clusters, which change according to the changes in the vector representations and numerical data of the clusters.

In the next piece of code, we are transforming 2-dimensional vectors to 21-dimensional to see more clarity in the clusters to find out the centers of the clusters

```
pca2 = PCA(n_components=3).fit(cluster_center)
pca2d = pca2.transform(cluster_center)
```

```
pca2d
```

```
array([[-0.72048798, -0.49547655,  0.18637994],
       [ 4.95311053, -0.07637796, -0.18538106],
       [-1.88803578, -0.92055054, -0.37924587],
       [-1.25843787, -0.75361297, -0.16498406],
       [-1.09506131,  2.43221488, -0.11232509],
       [ 0.0089124 , -0.18619686,  0.65555614]])
```

```
pca2d[:,0]
```

```
array([-0.72048798,  4.95311053, -1.88803578, -1.25843787, -1.09506131,
        0.0089124 ])
```

```
pca2d[:,1]
```

```
array([-0.49547655, -0.07637796, -0.92055054, -0.75361297,  2.43221488,
       -0.18619686])
```

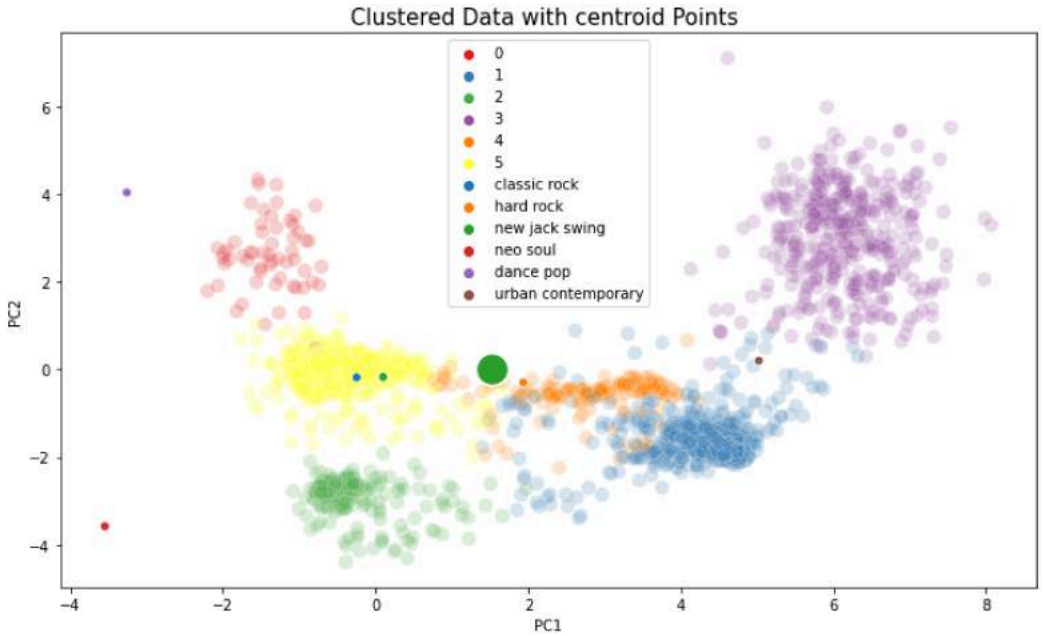
```
x1=[-0.24827018,  1.93369274,  0.09927498, -3.54708215, -3.2595292 ,
        5.0219138 ]
```

```
y1=[-0.18343355, -0.29577815, -0.17285663, -3.57870242,  4.03274577,
        0.19802499]
```

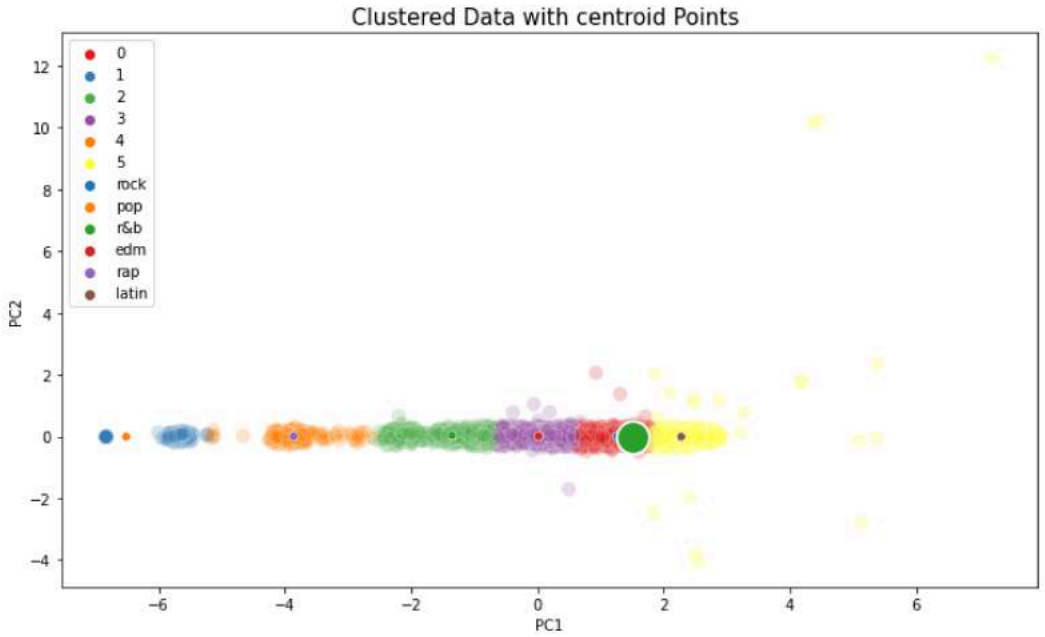
By converting the dimensions, we were able to reduce the dimensions from 2D to 21D with more clarity on the clusters.

This implies that dimensionality reduction in clustering analysis decreases computational costs by eliminating noisy data and improving genre classifications.

Our aim was to find out the centroids of the genres, and then to store those data in objects called X1 and Y1 that could be used to plot the results after dimensionality reduction, and see how it actually looks after dimensionality reduction as well.



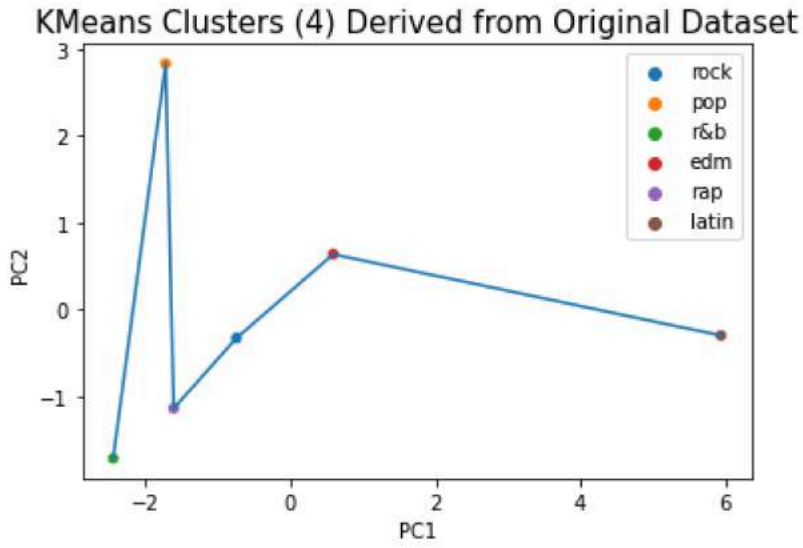
The green dot shows the new song cluster; the different colors indicate different subgenres, and the subgenres are also indicated by color. Since there are additional sub genres in the dataset that are only included in the Genres dataset, only the clustered formats will change.

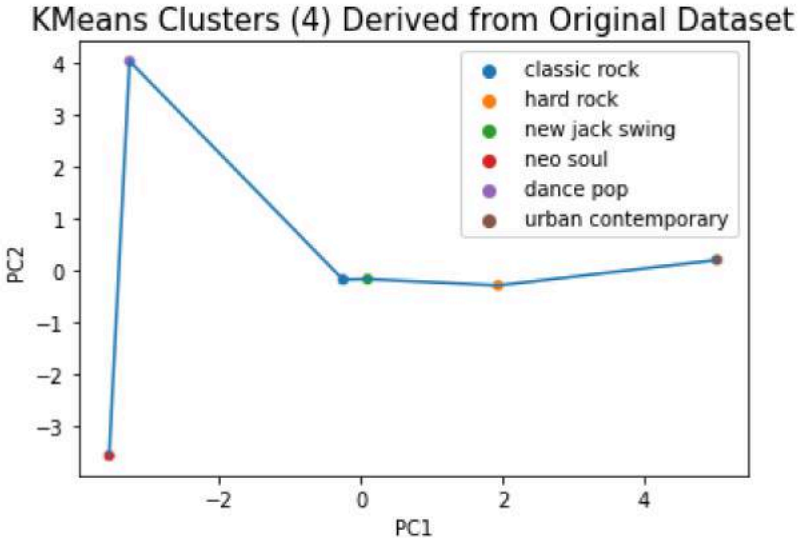


The data contains too many clusters for subgenres and too few for genres because we split them and attempted to analyze based on the dbow model for our thesis, as you can see in these plots of genres and subgenres, respectively.

```
#distance between consecutive centroids

for t in range(len(cluster_centers)-1):

    #print(cluster_centers[t])

    distance = np.sqrt(np.sum(np.array(cluster_centers[t]) -np.array(cluster_centers[t+1]))**2)
    print(distance)
```

```
4.554671160000001
4.9670060099999995
0.195289
1.8578624999999998
4.8089739
```





So after analyzing the clustered data now let's move on to find the actual distance between genres and subgenres and we can compare the results accordingly.
As I indicated in the main part at the beginning of the chapter, we are applying Euclidean distance using Numpy in the piece of code above.

As we know, machine learning algorithms that employ distance measurements often use the euclidean distance formula that is derived from Pythagoras' theorem. It provides me with the shortest distance between two adjacent places.

We may use a line chart to display the findings after selecting two centroid positions at random, at which time we can begin computing the distance between each one.

- For the time being, we'll take the Hard Rock to Classic Rock distance under the subgenre section to be 6.471. We also estimated distances for other genres and subgenres.
- And in the genre section if we see the distance between Rapp and Latin is around 4.8098.so this way we can count the distance between each genre and each sub-genres and then we can predict the songs which belong to which cluster.
- Therefore, we must determine how far the first centroid point is from the other centroid point.
- I've displayed the distance plots below in accordance with the outcomes that our model predicted.

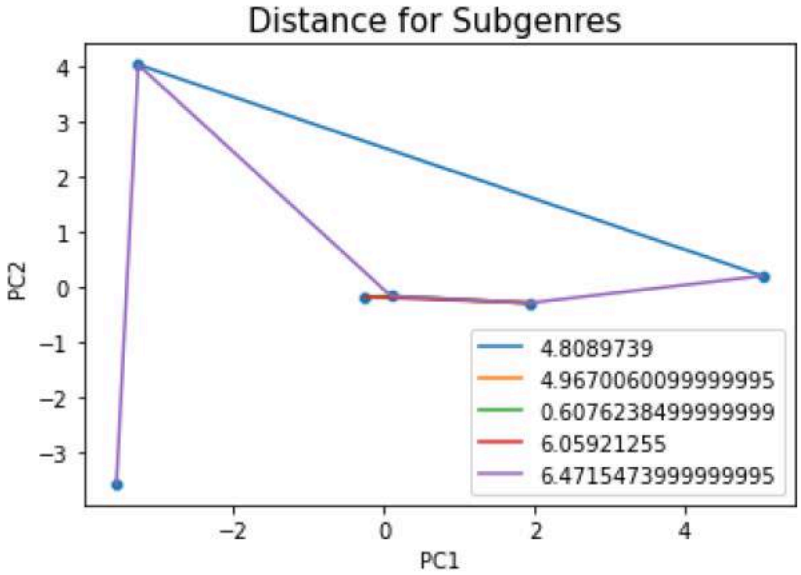
$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

We start by quickly reviewing the formula, which is really simple. We take the square root of the sum of squared differences in the constituents of two vectors, x and y.

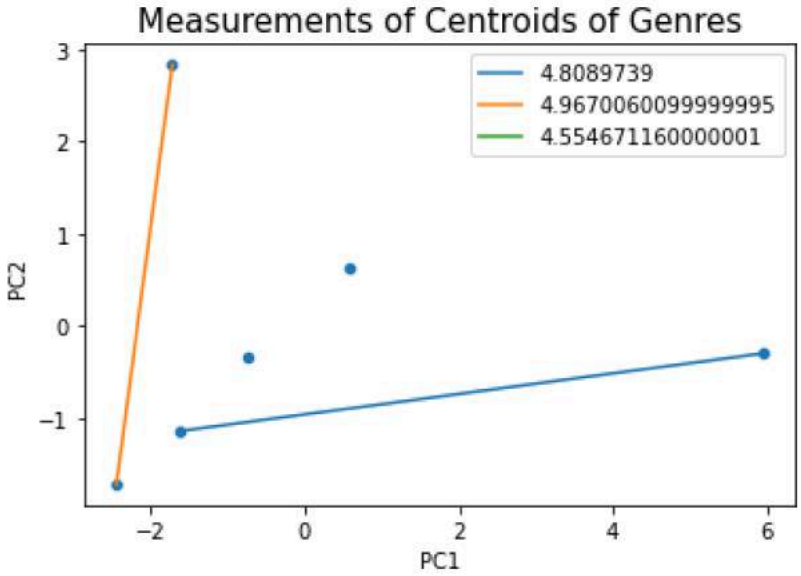
The method we are using to find out the distance is vectorized implementation method using the euclidean distance formula. I must admit that after reading a few study articles on this subject online, I was really optimistic about how well this strategy would work. Observing how Sklearn's euclidean distances performed also gave those aspirations a boost.

For smaller data samples, the technique is pretty similar to the cdist solution in terms of time, however, it doesn't scale well. For the biggest data sample, the for-loop technique takes almost as long as one without memory pre-allocation.

You can see the graph I created below, which I've plotted with each genre and each of its subgenres along with how far off they are from one another. and with this, a cluster analysis may be performed. Then we can determine which songs are related based on their genres. In the CSV file I made, the column appears as clusters on each row of songs.



The distance between a sub-genre centroid point and two sub-genre centroid points is shown in the above figure.



These measurements of centroid points of genres have been represented in this snapshot with color palettes and we can analyze them with previous results.

```
for t in range(len(averaged_cluster)-1):

    #print(cluster_centers[t])

    distance = np.sqrt(np.sum(np.array(averaged_cluster[t]) -np.array(averaged_cluster[t+1]))**2)
    print(distance)

```

```
7.621382684880552
4.901365106360334
1.3814119846030541
3.7967738644345395
6.270794164024314
```

In addition to finding the distance between clustered genres and sub-genres, I created an object that looked at the average distance between these categories.

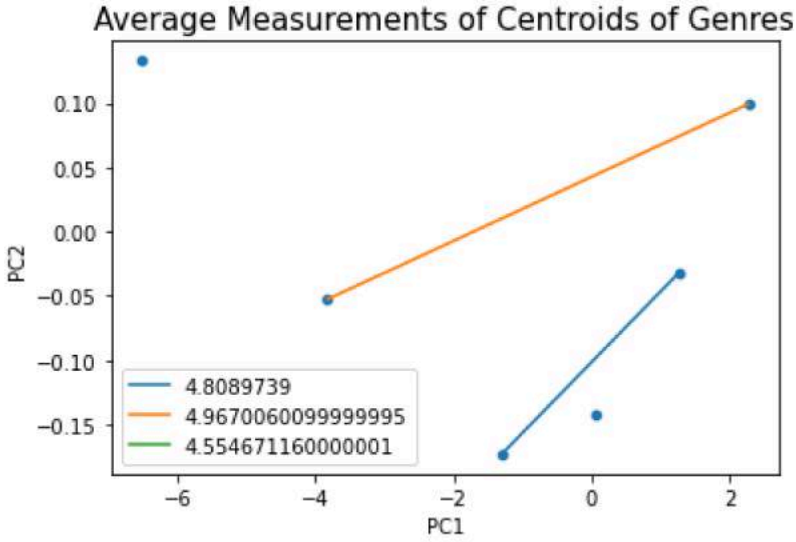
The purpose of this study will be to use lists and fit the PCA component transform into 2-dimensional space, then transform it and generate values, so I can list out the values after transformation.

As we got the values from the 3 lists we created through the PCA redundancy process, we will consider those values. The values' lengths are the same as the average cluster values.

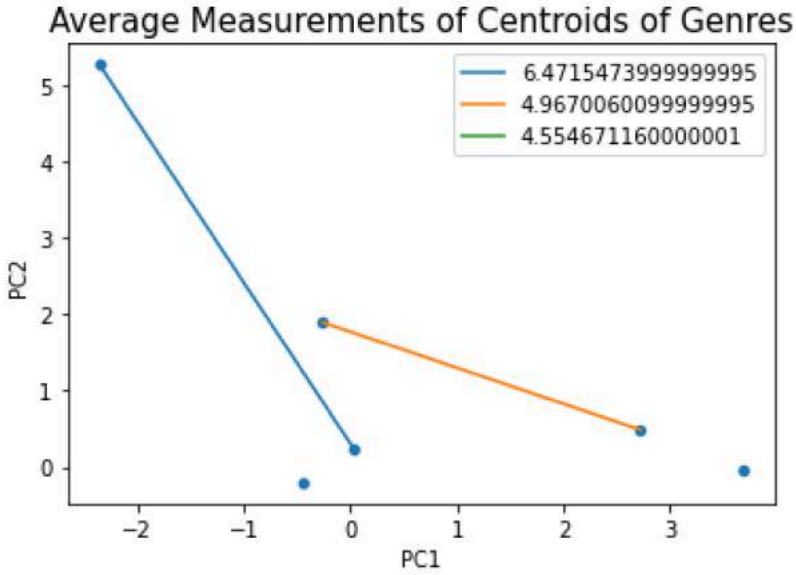
We have now acquired the average cluster values and combined them with the feature values we created earlier, generating the new label data frame.

Our average cluster was determined by this method, and we were also able to determine which song belonged to which cluster.

In Python, we created the same euclidean formula as can be seen in the above snapshot, and in the mathematical output, it shows the distance. For reference, you can also see the visualized output below the screenshots.



As You can see in our final Output we finally found out the distances between genres from one centroid point to another. These distances are considered to be average distances among the distance that we found in the previous section.



Here in this figure distances between Sub-Genres from one centroid point to another. These distances are considered to be average distances among the distance that we found in the previous section.

Chapter 5

Conclusion

5.1 Conclusion

In this Research work, I have discovered a wide variety of approaches during the period of my Thesis work, including several that I was not aware even available. Understanding the data and using it effectively went more or less according to plan.

Additionally, I learned new ideas from this research, including cluster analysis and Embeddings approaches. However, I found the practical portion to be the true challenge. We conducted a thorough quantitative investigation of word2vec embeddings for music genre recommendations in this research. I tried several various methods and ideas over the workflow for my thesis for a few months with the help of my Professors, but none seemed to be a Good Idea. Since they were not at all utilized for the final recommender, these failed attempts were not included in this project. However, I think it is a part of the Learning that I have acquired.

We found that it is tricky to outperform the performance of relatively basic models trained on song data on a variety of difficult classification tasks at the size of thousands of songs. Small averaging embeddings improvements to be feasible using sequence modeling, despite the fact that outcomes are Based on genres and cluster analysis, you can see which songs are similar to each other.

So this is how natural language processing is used to accomplish music genre cluster analysis. The task of classifying musical genres based on their resemblances to song genres is known as clustering.

In conclusion, I believe it is acceptable to say that I learned a lot and got the opportunity to become familiar with various deep learning techniques, which was my primary objective in the first place from this research I understood some basic ideas to implement them with my ideas. The pain I had at times while working on this project, in my opinion, was worthwhile. It is natural and commonly understood that there are times when an issue cannot be solved. But knowing something is not the same as experiencing it in your own inquiry, where you

have invested a lot of time and have certain expectations. Last but not least, I had never before taken part in a contest with these qualities. It has been really exciting and fulfilling to collaborate with a professor and professionals during these final weeks, meeting, discussing, testing new ideas, and enhancing the outcomes. It enables me to do this difficult assignment with a great feeling.

References

1) Baad, D. (2021, June 15). Implementing multi-class Text Classification with Doc2Vec. Medium. https://towardsdatascience.com/implementing-multi-class-text-classification-with-doc2vec-df7c3812824d
2) Bach, A. (2018). Treball final de grau WORD2VEC EMBEDDINGS FOR PLAYLST RECOMMENDATION. http://diposit.ub.edu/dspace/bitstream/2445/130481/3/memoria.pdf
3) Barla, N. (2022, July 21). The Ultimate Guide to Word Embeddings. Neptune.ai. https://neptune.ai/blog/word-embeddings-guide
4) Blogs, M. (2022, April 18). Music Recommendation System Revolutionizing the Music Streaming Industry. Muvi One. https://www.muvi.com/blogs/music-recommendation-system.html#:~:text=A%20music%20recommender%20solution%20is
5) Budhrani, A., Patel, A., & Ribadiya, S. (2020, November 1). Music2Vec: Music Genre Classification and Recommendation System. IEEE Xplore. https://doi.org/10.1109/ICECA49313.2020.9297559
6) Chakraborty, D. (2022a, July 4). Introduction to the bag-of-words (bow) model. PyImageSearch. https://pyimagesearch.com/2022/07/04/introduction-to-the-bag-of-words-bow-model/
7) Chakraborty, D. (2022b, July 11). Word2Vec: A study of embeddings in NLP. PyImageSearch. https://pyimagesearch.com/2022/07/11/word2vec-a-study-of-embeddings-in-nlp/

8) Çoban, Ö., & Karabey, I. (2017a, May 1). Music genre classification with word and document vectors. IEEE Xplore. https://doi.org/10.1109/SIU.2017.7960145
9) Çoban, Ö., & Karabey, I. (2017b, May 1). Music genre classification with word and document vectors. IEEE Xplore. https://doi.org/10.1109/SIU.2017.7960145
10) Ermolaev, A. (2018, August 25). Text clusterization using Python and Doc2vec. Medium. https://medium.com/@ermolushka/text-clusterization-using-python-and-doc2vec-8c499668fa61
11) Fessahaye, F., Perez, L., Zhan, T., Zhang, R., Fossier, C., Markarian, R., Chiu, C., Zhan, J., Gewali, L., & Oh, P. (2019). T-RECSYS: A Novel Music Recommendation System Using Deep Learning. IEEE International Conference on Consumer Electronics (ICCE). https://par.nsf.gov/servlets/purl/10088296
12) Gali, N., & Tiwari, D. V. (2021). Speech and Lyric-based Doc2Vec Music Recommendation System. International Journal of Engineering Research & Technology, 9(8). https://doi.org/10.17577/IJERTCONV9IS08015
13) GOYAL, C. (2021, June 21). Text Vectorization and Word Embedding | Guide to Master NLP (Part 5). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/part-5-step-by-step-guide-to-master-nlp-text-vectorization-approaches/
14) Grau D', & Bach, A. (2018). Treball final de grau word2vec embeddings for playlist recommendation. http://diposit.ub.edu/dspace/bitstream/2445/130481/3/memoria.pdf
15) Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., & Ng, A. Y. (2014). Deep Speech: Scaling up end-to-end Speech Recognition. ArXiv:1412.5567 [Cs]. https://arxiv.org/abs/1412.5567

16) He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural Collaborative Filtering. ArXiv:1708.05031 [Cs], 978-1-6654-7200-5. https://arxiv.org/abs/1708.05031
17) History-biography. (2019, December 3). Spotify. History and Biography. https://history-biography.com/spotify/
18) https://kodzilla.pl, & dev@kodzilla.pl. (2020, July 21). Addepto. Addepto. https://addepto.com/blog/deep-learning-architecture/
19) Huilgol, P. (2020, February 27). BoW Model and TF-IDF for Creating Feature from Text. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/
20) IBM. (n.d.). What is Natural Language Processing? | IBM. Www.ibm.com. https://www.ibm.com/topics/natural-language-processing
21) Kapl, D., coding, ylan K. has years of experience as a S. D. S. H. enjoys, teaching, & everyone, has created this website to make M. L. accessible to. (2022, October 6). Machine Learning 101: Count Vectorizer Vs TFIDF Vectorizer EML. Enjoymachinelearning.com. https://enjoymachinelearning.com/blog/countvectorizer-vs-tfidfvectorizer/
22) Kharwal, A. (2022, April 5). Clustering music genres with machine learning | aman kharwal. Thecleverprogrammer. https://thecleverprogrammer.com/2022/04/05/clustering-music-genres-with-machine-learning/
23) Kim, D., Seo, D., Cho, S., & Kang, P. (2019). Multi-co-training for document classification using various document representations: TF–IDF, LDA, and Doc2Vec. Information Sciences, 477, 15–29. https://doi.org/10.1016/j.ins.2018.10.006

24) Ku, B. W., Schuman, C. D., Adnan, M. M., Mintz, T. M., Pooser, R., Hamilton, K. E., Rose, G. S., & Lim, S. K. (2022). Unsupervised Digit Recognition Using Cosine Similarity in a Neuromemristive Competitive Learning System. ACM Journal on Emerging Technologies in Computing Systems, 18(2), 1–20. https://doi.org/10.1145/3473036
25) Kumar, A., Rajpal, A., & Rathore, D. (2018). Genre Classification using Word Embeddings and Deep Learning. 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI). https://doi.org/10.1109/icacci.2018.8554816
26) Lau, J. H., & Baldwin, T. (2016). An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. ArXiv:1607.05368 [Cs]. https://arxiv.org/abs/1607.05368
27) Le, Q., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. https://arxiv.org/pdf/1405.4053.pdf
28) Lin, Y. (2022, October 1). The Development of Music Recommendation Systems Based on Song Characteristics. IEEE Xplore. https://doi.org/10.1109/ICDSCA56264.2022.9987860
29) Mani, K., Verma, I., Meisheri, H., & Dey, L. (2018). Multi-Document Summarization Using Distributed Bag-of-Words Model. ArXiv:1710.02745 [Cs]. https://arxiv.org/abs/1710.02745
30) Nicholson, C. (n.d.). A beginner's guide to word2vec and neural word embeddings. Pathmind; Chris Nicholson. https://wiki.pathmind.com/word2vec
31) Niyazov, A., Mikhailova, E., & Egorova, O. (2021a, May 1). Content-based Music Recommendation System. IEEE Xplore. https://doi.org/10.23919/FRUCT52173.2021.9435533

32) Niyazov, A., Mikhailova, E., & Egorova, O. (2021b, May 1). Content-based Music Recommendation System. IEEE Xplore. https://doi.org/10.23919/FRUCT52173.2021.9435533
33) Pastukhov, D. (2022, February 9). Inside Spotify's Recommender System: A Complete Guide to Spotify Recommendation Algorithms. Www.music-Tomorrow.com. https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022
34) Patra, B. G., Das, D., & Bandyopadhyay, S. (2017, December 1). Retrieving Similar Lyrics for Music Recommendation System. ACLWeb; NLP Association of India. https://aclanthology.org/W17-7536/
35) Patra, B., Das, D., & Bandyopadhyay, S. (2015). Mood Classification of Hindi Songs based on Lyrics. In NLP Association of India (pp. 261–267). https://aclanthology.org/W15-5939.pdf
36) Prabhakaran, S. (2018, October 16). Gensim tutorial - A complete beginners guide. Machine Learning Plus. https://www.machinelearningplus.com/nlp/gensim-tutorial/
37) Pratim Barman, M., Dahekar, K., Anshuman, A., & Awekar, A. (n.d.). It's Only Words And Words Are All I Have. Retrieved February 6, 2023, from https://arxiv.org/pdf/1901.05227.pdf
38) Recurrent Neural Network (RNN) Tutorial: Types and Examples [Updated] | Simplilearn. (n.d.). Simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn#what_is_a_recurrent_neural_network_rnn

39) Reevesman, A. (2020, July 19). Lyric-based Song Recommendation with Doc2Vec Embeddings and Spotify's API. Medium. https://towardsdatascience.com/lyric-based-song-recommendation-with-doc2vec-embeddings-and-spotifys-api-5a61c39f1ce2
40) Ricci, F., Lior Rokach, Shapira, B., & Kantor, P. B. (2010). Recommender Systems Handbook. Springer Science & Business Media.
41) Ripenko, S., & Hasiuk, N. (2022 10). ELIFTECH | Blog | All you need to know about a music recommendation system with a step-by-step guide to creating it. ElifTech Insights. https://www.eliftech.com/insights/all-you-need-to-know-about-a-music-recommendation-system-with-a-step-by-step-guide-to-creating-it/
42) Sachi Nandan Mohanty, Jyotir Moy Chatterjee, Jain, S., Elngar, A. A., & Gupta, P. (2020). Recommender System with Machine Learning and Artificial Intelligence. John Wiley & Sons.
43) Schedl, M., Knees, P., McFee, B., Bogdanov, D., & Kaminskas, M. (2015a). Music Recommender Systems. Recommender Systems Handbook, 453–492. https://doi.org/10.1007/978-1-4899-7637-6_13
44) Schedl, M., Knees, P., McFee, B., Bogdanov, D., & Kaminskas, M. (2015b). Music Recommender Systems. Recommender Systems Handbook, 453–492. https://doi.org/10.1007/978-1-4899-7637-6_13
45) Sen, I. (2018, May 22). How AI helps Spotify win in the music streaming world - Outside Insight. Outside Insight. https://outsideinsight.com/insights/how-ai-helps-spotify-win-in-the-music-streaming-world/

46) Simeon Kostadinov. (2017, December 2). How Recurrent Neural Networks work. Medium; Towards Data Science. https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

47) Stern, S. (2021). Analysis of music genre clustering algorithms analysis of music genre clustering algorithms. https://dc.uwm.edu/cgi/viewcontent.cgi?article=3844&context=etd

48) Team, P. (2021, September 9). Song2Vec: Music Recommender. Algorithm Data Science School. https://algoritmaonline.com/song2vec-music-recommender/

49) Techlabs, M. (2021, August 18). What Are the Types of Recommendation Systems? MLearning.ai. https://medium.com/mlearning-ai/what-are-the-types-of-recommendation-systems-3487cbafa7c9

50) Verma, Y. (2022a, February 20). How to implement a Doc2Vec model using Gensim? Analytics India Magazine. https://analyticsindiamag.com/how-to-implement-a-doc2vec-model-using-gensim/

51) Verma, Y. (2022b, February 22). A Guide to Document Embeddings Using Distributed Bag-of-Words (DBOW) Model. Analytics India Magazine. https://analyticsindiamag.com/a-guide-to-document-embeddings-using-distributed-bag-of-words-dbow-model/

52) Wang, D., Liang, Y., Xu, D., Feng, X., & Guan, R. (2018). A content-based recommender system for computer science publications. Knowledge-Based Systems, 157, 1–9. https://doi.org/10.1016/j.knosys.2018.05.001

53) What are Recurrent Neural Networks? | IBM. (n.d.). Www.ibm.com. https://www.ibm.com/topics/recurrent-neural-networks

54) Zhang, Y., Jin, R., & Zhou, Z.-H. (2010). Understanding bag-of-words model: a Statistical Framework. International Journal of Machine Learning and Cybernetics, 1(1-4), 43–52. https://doi.org/10.1007/s13042-010-0001-0