**CAPSTONE PROJECT REPORT**



**Project**

Abstract NLP

**Guided by Professor**

Denis Vrdoljak

**Submitted By**

Mytreyi Reddy

Pooja Varadaraj

Kavitha Kallakere Indrakumar

DeivaSubhaRanjani Pandurangan Ramamurthy

**Table of Contents**

[Abstract concept extraction of large datasets using NLP 3](#_Toc58361545)

[Problem Statement 4](#_Toc58361546)

[Objectives/Scope 5](#_Toc58361547)

[Business Requirements: 5](#_Toc58361548)

[Functional Requirements: 5](#_Toc58361549)

[NLP standard practices: 6](#_Toc58361550)

[Tools and Techniques used in NLP project implementation: 7](#_Toc58361551)

[Technique 1: Gensim Library -Doc2vec 7](#_Toc58361552)

[Technique 2: Topic Modelling 12](#_Toc58361553)

[Additional Research Methodologies/Techniques: 14](#_Toc58361554)

[Technique 1: Word2Vec model implementation using TensorFlow 14](#_Toc58361555)

[Technique 2: Text Embeddings 15](#_Toc58361556)

[Scoring Metrics used in our project: 18](#_Toc58361557)

[Heatmap Analysis: 20](#_Toc58361558)

[Hyperparameter Tuning Techniques for the Model: 21](#_Toc58361559)

[Grid Search 22](#_Toc58361560)

[Random Search 24](#_Toc58361561)

[Conclusion: 29](#_Toc58361563)

[Appendix: 29](#_Toc58361564)

[i. Success metrics: 30](#_Toc58361565)

[ii.Team Weekly Deliverables 30](#_Toc58361566)

[iii.Team Weekly Meeting Schedule: 31](#_Toc58361567)

[iv.Overall Challenges faced in the project: 32](#_Toc58361568)

[v. Deliverables 32](#_Toc58361569)

[vi. Our Team: 33](#_Toc58361570)

# **Abstract concept extraction of large datasets using NLP**

NLP is the ability to extract and act upon information from natural languages in a way meaningful to applications and machine learning systems.[[1]](#footnote-1)

Concepts are sequences of words that represent real or imaginary entities or ideas that users are interested in. Abstract Concept extraction, a subdomain of natural language processing (NLP) with a focus on extracting concepts of interest, has been adopted to computationally extract information from text for a wide range of applications in the past few years.

The abstract concept extraction can be classified based on:

1. Macro concept extraction– provides a general understanding of the document as a whole.

* Typically performed with statistical techniques
* It is used for: clustering, categorization, similarity, topic analysis, word clouds, and summarization

1. Micro concept extraction– extracts understanding from individual phrases or sentences.

* Typically performed with NLP techniques
* It is used for: extracting facts, entities (see above), entity relationships, actions, and metadata fields

Building a web of concepts will form the backbone of the next generation of search technology and it will also help in analyzing large datasets easily. A web of concepts would not only allow us to identify user intent better, but also to rank content better, support more expressive queries and present the integrated information better. Our definition of a concept is based on its usefulness to people. That is, a string is a concept if a “significant” number of people say it represents an entity, event or topic known to them. For instance, “Flying Pigs Shoe Store” is not a concept if only one or two people know about this store, even though this store may have a web page where “Flying Pigs Shoe Store” appears.[[2]](#footnote-2)

Once we have built the pipeline, there are multiple approaches to automate this extraction process. They are:

1. **Rule-based Approach**: We define a set of rules for the syntax and other grammatical properties of a natural language and then use these rules to extract information from text
2. **Supervised:** Let’s say we have a sentence S. It has two entities E1 and E2. Now, the supervised machine learning model has to detect whether there is any relation (R) between E1 and E2. So, in a supervised approach, the task of relation extraction turns into the task of relation detection. The only drawback of this approach is that it needs a lot of labeled data to train a model
3. **Semi-supervised:** When we don’t have enough labeled data, we can use a set of seed examples (triples) to formulate high-precision patterns that can be used to extract more relations from the text

This capstone will build analytics models to characterize the COVID-19 outbreak. We will initially collect the COVID data and implement the NLP (Natural Language Processing), Machine learning algorithms to build a model that can input any dataset and gives the characteristics of the data by performing the analysis.

We would like to extract information from the corona data sources starting with standard techniques like bag of words concept, vectorization techniques and document similarity scores then proceed with compressed vectors created by encoders to create the abstract of any dataset. This can be used in decreasing the defects and increasing the code quality in the data pipelines.

# **Problem Statement**

The recent corona outbreak gave the Data enthusiast the real time challenge. To come up with the analysis within the shortest time, to understand each and every dataset, choosing the right techniques for the selected dataset is always a challenge, given the large amount of data our industries used. In the world of Big data, effective concept extraction using Natural language processing (NLP) to extract information for knowledge graphs is identified as a problem. We would like to take this project to build a model that takes any dataset as an input to build a knowledge graph which is incorporated into improving the code quality and decreasing defects in the development pipelines. We would start with the corona datasets to identify the outbreak and analyze the different aspects and measures.

# **Objectives/Scope**

The Objective of the capstone project is to build a data science pipeline that takes corpus of texts as input, concept and automates the vectorization model. The model can take in any abstract data and convert them into vectors. The model is to be optimized using Hyper Parameter Tuning Concepts like grid search and random search. Also, to build a cost function which would be able to determine the accuracy of the model.

To analyze the free text engineers, spend a lot of time vectorizing the text and this consumes a lot of bandwidth and slows the process of execution. To overcome this model will act as a plugin for any NLP process which is automated to do vectorization. This model can be used predominantly in building Chatbots, based on the vectorized training data, it would have the ability to interpret the text.

## **Business Requirements:**

1. The System will provide a semi-automated system that can create the text vectorization.
2. The Model must be domain independent and should provide the initial machine learning model on text data.
3. The Model will also create a pipeline for building text vectorization.

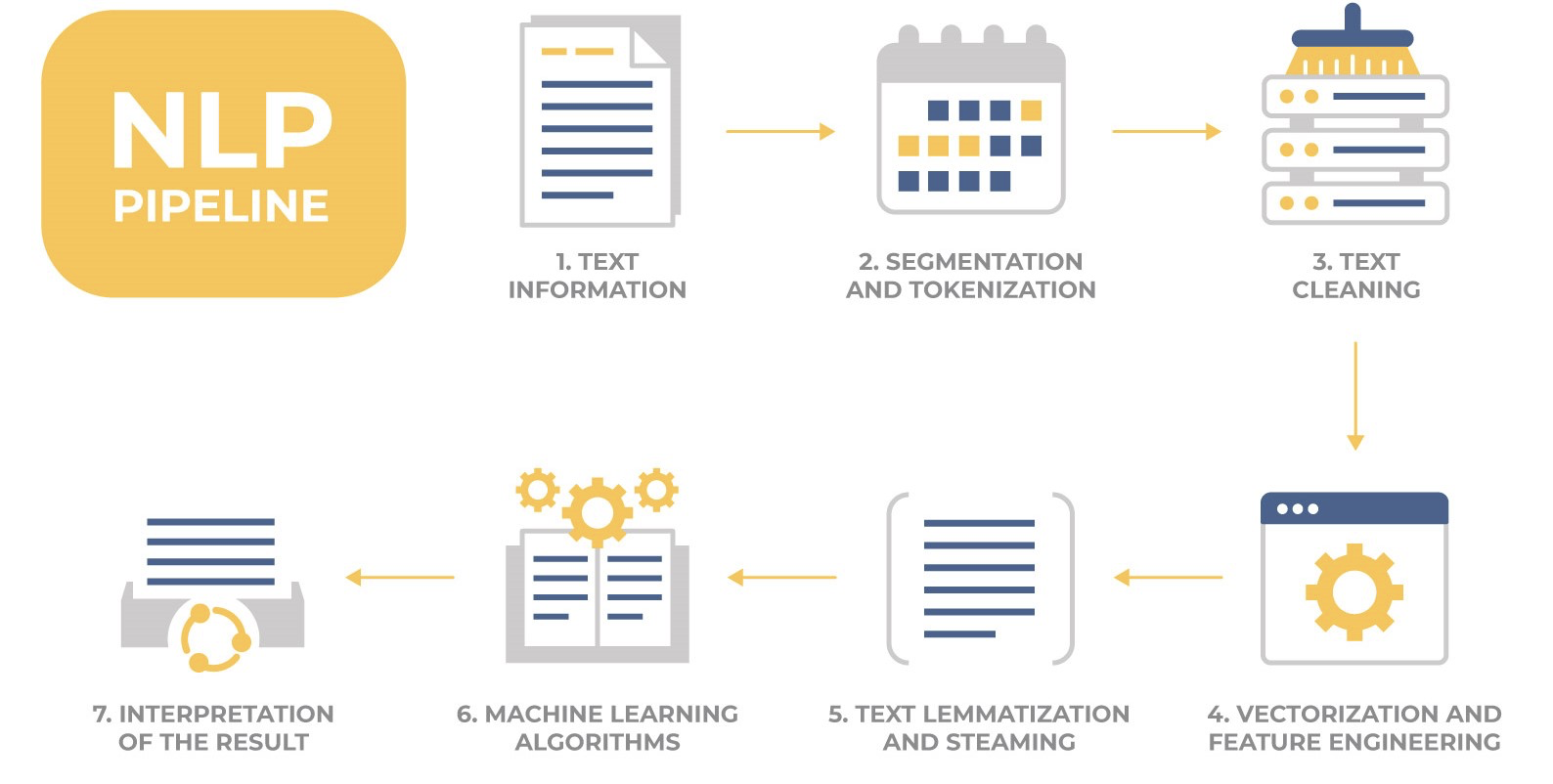
## **Functional Requirements:**

1. The model will take the ability to take any dataset as the input.
2. The Model will vectorize the input data.
3. The model will compile all the documents, feature them and compare their features.
4. The model will use compressed vectors created by the encoder
5. The model will be used to develop the knowledge graphs based on the data input.

# **NLP standard practices:**

Typically, any NLP-based problem can be solved by a methodical workflow that has a sequence of steps.We usually start with a corpus of text documents and follow standard processes of text wrangling and pre-processing, parsing and basic exploratory data analysis. Based on the initial insights, we usually represent the text using relevant feature engineering techniques. Depending on the problem at hand, we either focus on building predictive supervised models or unsupervised models, which usually focus more on pattern mining and grouping. Finally, we evaluate the model and the overall success criteria with relevant stakeholders or customers, and deploy the final model for future usage.[[3]](#footnote-3)

1. Text Information
2. Segmentation and tokenization
3. Text Cleaning
4. Vectorization and Feature Engineering
5. Text Lemmatization and Stemming
6. NLP Libraries/ Machine Learning Algorithms
7. Interpretation of the results.

**[[4]](#footnote-4)**

# **Tools and Techniques used in NLP project implementation:**

## **Technique 1: Gensim Library -Doc2vec**

Gensim = “Generate Similar” is a popular open source natural language processing (NLP) library used for unsupervised topic modeling. It uses top academic models and modern statistical machine learning to perform various complex tasks such as −

* Building document or word vectors
* Corpora
* Performing topic identification
* Performing document comparison (retrieving semantically similar documents)
* Analyzing plain-text documents for semantic structure

**Doc2Vec Implementation:**

Doc2Vec model is an extension to the Word2Vec model which was created and published in 2013 by a team of researchers led by [Tomas Mikolov](https://en.wikipedia.org/wiki/Tomas_Mikolov) at [Google](https://en.wikipedia.org/wiki/Google). It is used to create a vector representation of a group of words taken collectively as a single unit. It doesn’t only give the simple average of the words in the sentence.

**Brief on word2vec[[5]](#footnote-5):**

Since doc2vec is an extension of word2vec it would be good to have a general understanding of it as well.

**Word2vec** is a technique for [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing). The word2vec algorithm uses a [neural network](https://en.wikipedia.org/wiki/Neural_network) model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a [vector](https://en.wikipedia.org/wiki/Vector_(geometry)). The vectors are chosen carefully such that a simple mathematical function (the [cosine similarity](https://en.wikipedia.org/wiki/Cosine_similarity) between the vectors) indicates the level of [semantic similarity](https://en.wikipedia.org/wiki/Semantic_similarity) between the words represented by those vectors.

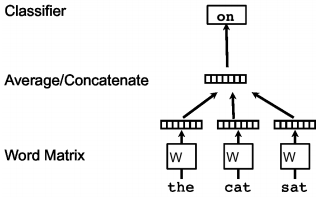


Figure 1. A framework for learning word vectors. Context of three words (“the,” “cat,” and “sat”) is used to predict the fourth word (“on”). The input words are mapped to columns of the matrix W to predict the output word

Word2vec can utilize either of two model architectures to produce a [distributed representation](https://en.wikipedia.org/wiki/Distributed_representation) of words: [continuous bag-of-words](https://en.wikipedia.org/wiki/Continuous_bag-of-words) (CBOW) or continuous [skip-gram](https://en.wikipedia.org/wiki/Skip-gram).

**Continuous bag of words -** Continuous bag of words creates a sliding window around the current word, to predict it from “context” — the surrounding words. Each word is represented as a feature vector. After training, these vectors become the word vectors.

**Skip gram -** The second algorithm is actually the opposite of CBOW: instead of predicting one word each time, we use 1 word to predict all surrounding words (“context”). **Skip gram**is much slower than CBOW, but considered more accurate with infrequent words.

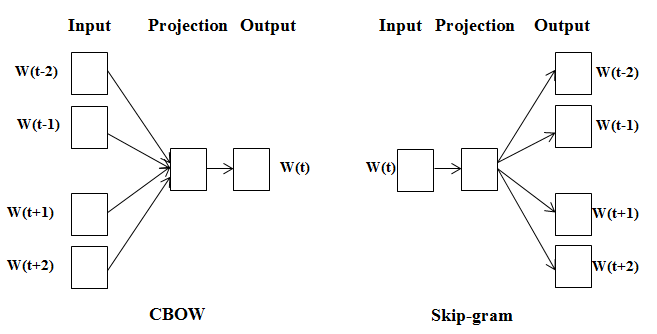


Figure 2: Schematic representation of CBOW and Skip-gram as explained above

Words maintain logical (grammatical) structure but documents don’t have any logical structures. To solve this problem another vector (Paragraph ID) needs to be added with the word2vec model to get two variations of doc2vec are available:

1. **Distributed Memory Model of Paragraph Vectors (PV-DM)** - Distributed Memory (DM) model is similar to Continuous-Bag-of-Words (CBOW) model in word2vec which attempts to guess the output (target word) from its neighboring words (context words) with the addition of a paragraph ID.

In our Paragraph Vector framework (see Figure 3), every paragraph is mapped to a unique vector, represented by a column in matrix D and every word is also mapped to a unique vector, represented by a column in matrix W. The paragraph vector and word vectors are averaged or concatenated to predict the next word in a context. In the experiments, we use concatenation as the method to combine the vectors.

The paragraph token can be thought of as another word. It acts as a memory that remembers what is missing from the current context – or the topic of the paragraph. For this reason, we often call this model the Distributed Memory Model of Paragraph Vectors (PV-DM).

In summary, the algorithm itself has two key stages: 1) training to get word vectors W, and 2) “the inference stage” to get paragraph vectors D for new paragraphs

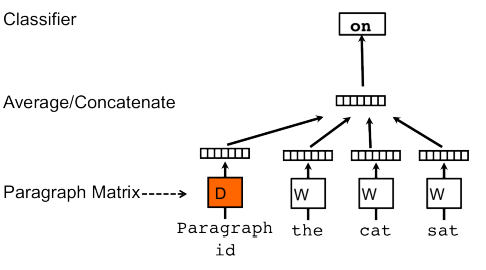


Figure 3. A framework for learning paragraph vectors. This framework is similar to the framework presented in Figure 1; the only change is the additional paragraph token that is mapped to a vector via matrix D. In this model, the concatenation or average of this vector with a context of three words is used to predict the fourth word. The paragraph vector represents the missing information from the current context and can act as a memory of the topic of the paragraph.

1. **Distributed Bag of Words version of Paragraph Vector (PV-DBOW) -** Distributed Bag-Of-Words (DBOW) Model similar to skip-gram model of word2vec, which guesses the context words from a target word. The only difference between skip-gram and distributed bag of words (DBOW) is instead of using the target word as the input, Distributed Bag of Words (DBOW) takes the document ID (Paragraph ID) as the input and tries to predict randomly sampled words from the document. The above method considers the concatenation of the paragraph vector with the word vectors to predict the next word in a text window.

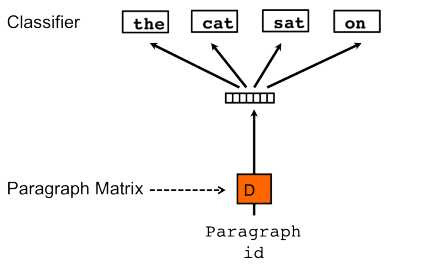


Figure 3. Distributed Bag of Words version of paragraph vectors. In this version, the paragraph vector is trained to predict the words in a small window

For our project we have chosen the PV-DM model because it preserves the word order in a document whereas PV-DBOW just uses the bag of words approach, which doesn’t preserve any word order**.**



Explanation on the parameters used in our model:

* epochs (int, optional) – Number of iterations (epochs) over the corpus. Defaults to 10 for Doc2Vec
* dm ({1,0}, optional) – Defines the training algorithm. If dm=1, ‘distributed memory’ (PV-DM) is used. Otherwise, a distributed bag of words (PV-DBOW) is employed.
* vector\_size (int, optional) – Dimensionality of the feature vectors.
* window (int, optional) – The maximum distance between the current and predicted word within a sentence.
* alpha (float, optional) – The initial learning rate.
* min\_alpha (float, optional) – Learning rate will linearly drop to min\_alpha as training progresses.
* min\_count (int, optional) – Ignores all words with total frequency lower than this.

**Steps followed in our Project for Doc2Vec Implementation:**

1. We tagged the data using the lemmitized column.
2. Trained the model using the tagged data.
3. Loaded the model.
4. Saved the model.
5. Using the infer vector on the saved model we convert any string input into a vector.
6. Using cosine similarity, we find the similarity between the different strings passed based on our trained model.

## **Technique 2: Topic Modelling[[6]](#footnote-6)**

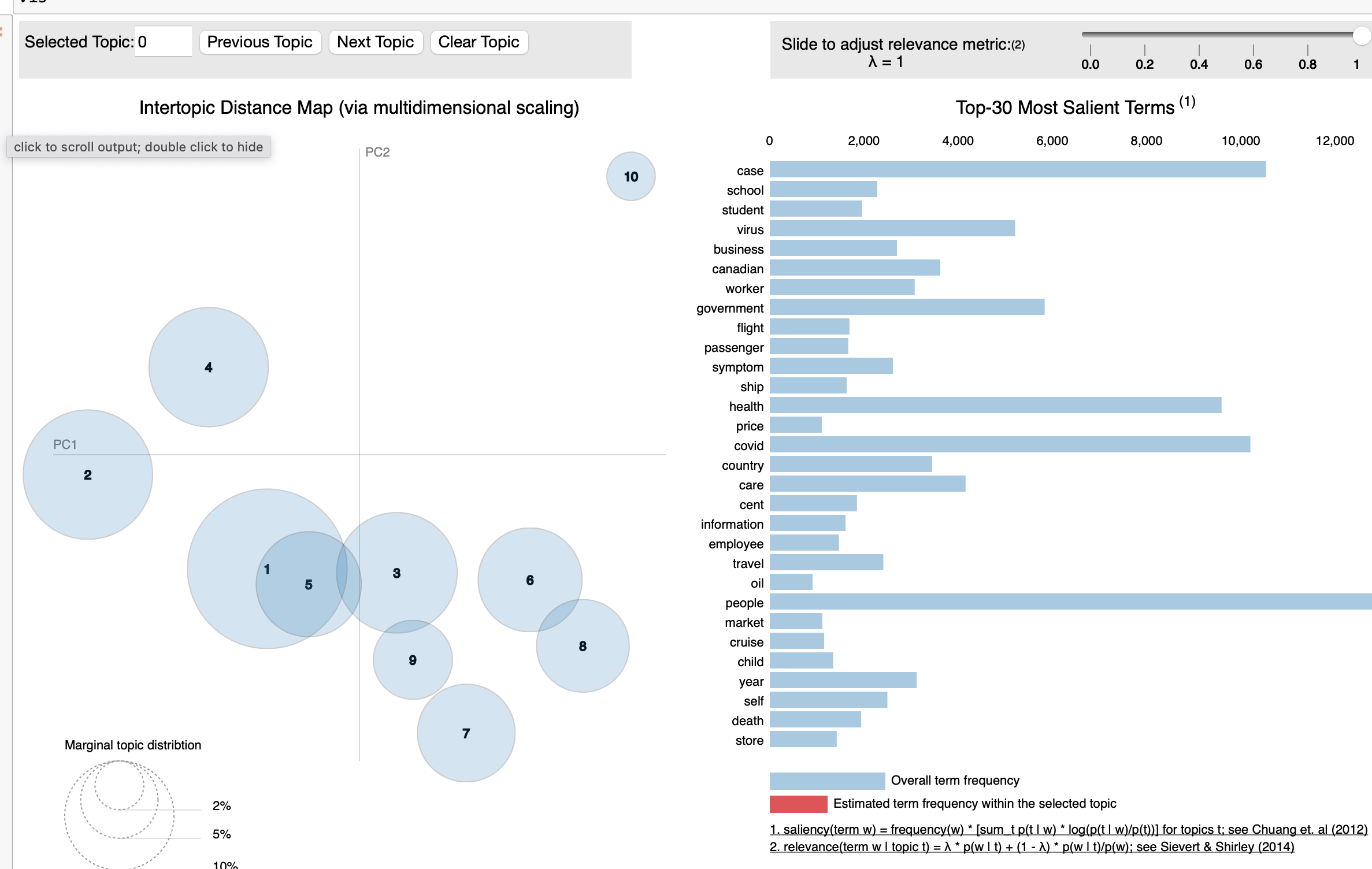
In machine learning and natural language processing, a topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. Topic modeling is a frequently used text-mining tool for discovery of hidden semantic structures in a text body. It is an unsupervised approach used for finding and observing the bunch of words (called “topics”) in large clusters of texts. Topics can be defined as “a repeating pattern of co-occurring terms in a corpus”. Topic Models are very useful for the purpose for document clustering, organizing large blocks of textual data, information retrieval from unstructured text and feature selection.[[7]](#footnote-7)

We are using Latent Dirichlet Allocation which is the most popular topic modeling technique.

LDA assumes documents are produced from a mixture of topics. Those topics then generate words based on their probability distribution. Given a dataset of documents, LDA backtracks and tries to figure out what topics would create those documents in the first place.

Parameters of LDA

* Alpha and Beta Hyperparameters – alpha represents document-topic density and Beta represents topic-word density. Higher the value of alpha, documents are composed of more topics and lower the value of alpha, documents contain fewer topics. On the other hand, higher the beta, topics are composed of a large number of words in the corpus, and with the lower value of beta, they are composed of few words.
* Number of Topics – Number of topics to be extracted from the corpus.
* Number of Topic Terms – Number of terms composed in a single topic. It is generally decided according to the requirement. If the problem statement talks about extracting themes or concepts, it is recommended to choose a higher number, if the problem statement talks about extracting features or terms, a low number is recommended.
* Number of Iterations / passes – Maximum number of iterations allowed to LDA algorithm for convergence.[[8]](#footnote-8)



## 

**How to infer pyLDAvis’s output?[[9]](#footnote-9)**

Each bubble on the left-hand side plot represents a topic. The larger the bubble, the more prevalent is that topic. A good topic model will have fairly big, non-overlapping bubbles scattered throughout the chart instead of being clustered in one quadrant. A model with too many topics, will typically have many overlaps, small sized bubbles clustered in one region of the chart. If you move the cursor over one of the bubbles, the words and bars on the right-hand side will update. These words are the salient keywords that form the selected topic.

## **Additional Research Methodologies/Techniques:**

## **Technique 1: Word2Vec model implementation using TensorFlow**

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

How word2vec works:

1. The idea behind word2vec is that:
2. Take a 3-layer neural network. (1 input layer + 1 hidden layer + 1 output layer)
3. Feed it a word and train it to predict its neighboring word.
4. Remove the last (output layer) and keep the input and hidden layer.

Now, input a word from within the vocabulary. The output given at the hidden layer is the ‘word embedding’ of the input word.

High-level steps followed in implementing tensor flow:

1. We need to convert the input into an output pair such that if we input a word, it should predict the neighboring words: the n words before and after it, where n is the parameter window size.
2. We created a dictionary which translates words to integers and integers to words.
3. Listed our sentences as a list of words.
4. Generate our training data. This basically gives a list of word, word pairs. (we are considering a window size of 2)
5. We convert our training data into one hot vector.
6. We created our tensor flow model to convert the training data into embedded representations.
7. We have the embedded dimension and make a prediction about the neighbor. To make the prediction we used SoftMax.
8. Now we train our model.
9. We tried to modify our W1 and B1 values to get the least loss value.

We implemented the tensor flow in AWS Sagemaker since the computation time required on our local systems was too long.

## **Technique 2: Text Embeddings**

Text Embeddings are real valued vector representations of strings. We build a dense vector for each word, chosen so that it’s similar to vectors of words that appear in similar contexts. Word embeddings are considered a great starting point for most deep NLP tasks. We are using Skip gram and CBoW for Word2vec:

**Skip-Gram [[10]](#footnote-10):** Skip-gram learns to predict the context words from a given word, in case where two words (one appearing infrequently and the other more frequently).

SoftMax Objective: Skip-gram’s objective is to predict the contexts of a given target-word. The contexts are immediate neighbors of the target and are retrieved using a window of an arbitrary size *n* — by capturing *n* words to the left of the target and *n* words to its right.

Architecture: Skip-gram is a simple neural network with only one hidden layer. The input to the network is a one-hot encoded vector representation of a target-word — all of its dimensions are set to zero, apart from the dimension corresponding to the target-word. The output is the probability distribution over all words in the vocabulary which defines the likelihood of a word being selected as the input word’s context

Steps:

1. NLP process - cleaning and tokenize
2. Word2Vec constructor takes a broad range of parameters, but we will only concentrate on a few that are most relevant:
   1. **sentences** — The iterate able over the tokenized sentences we will train on (the Brown sentences).
   2. **size** — The dimensionality of our embeddings.
   3. **window** — This determines which words are considered contexts of the target. For the window of size *n* the contexts are defined by capturing n words to the left of the target and n words to its right.
   4. **min\_count** — We can use this parameter to tell the model to ignore some infrequent words — don’t create an embedding for them and don’t include them as contexts.
   5. **negative** — Defines the number of negative samples (incorrect training pair instances) that are drawn for each good sample (see the Skip-gram section).
   6. **iter** — How many epochs do we want to train for — how many times we want to pass through our training data.
   7. **workers** — Determines how many worker threads will be used to train the model.
3. Define and Train the Model
4. Evaluating the Model
5. Building a Context Dependent Model

**Advantages of Skip-Gram Model**

1. Skip-gram model can capture two semantics for a single word. i.e it will have two vector representations of Apple. One for the company and other for the fruit.
2. Skip-gram with negative sub-sampling outperforms every other method generally.

**Continuous Bag of Words** [[11]](#footnote-11): The way CBOW works is that it tends to predict the probability of a word given a context. A context may be a single word or a group of words. But for simplicity, I will take a single context word and try to predict a single target word.

Suppose, we have a corpus C = “Hey, this is a sample corpus using only one context word.” and we have defined a context window of 1.

The flow is as follows:

1. The input layer and the target, both are one- hot encoded of size [1 X V]. Here V=10 in the above example.
2. There are two sets of weights. one is between the input and the hidden layer and second between the hidden and output layer.  
   Input-Hidden layer matrix size = [V X N], hidden-Output layer matrix size = [N X V], Where N is the number of dimensions, we choose to represent our word in. It is arbitrary and a hyper-parameter for a Neural Network. Also, N is the number of neurons in the hidden layer. Here, N=4.
3. There is a no activation function between any layers. (More specifically, I am referring to linear activation)
4. The input is multiplied by the input-hidden weights and called hidden activation. It is simply the corresponding row in the input-hidden matrix copied.
5. The hidden input gets multiplied by hidden- output weights and output is calculated.
6. Error between output and target is calculated and propagated back to re-adjust the weights.
7. The weight between the hidden layer and the output layer is taken as the word vector representation of the word.

**Advantages of CBOW:**

1. Being probabilistic is nature, it is supposed to perform superior to deterministic methods(generally).
2. It is low on memory. It does not need to have huge RAM requirements like that of a co-occurrence matrix where it needs to store three huge matrices.

**Disadvantages of CBOW:**

1. CBOW takes the average of the context of a word (as seen above in calculation of hidden activation). For example, Apple can be both a fruit and a company but CBOW takes an average of both the contexts and places it in between a cluster for fruits and companies.
2. Training a CBOW from scratch can take forever if not properly optimized.

# **Scoring Metrics used in our project:**

1. **Cosine Similarity:** Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity and larger the angle the lesser is the similarity between the strings taken as input.
2. **Cost Function:** It is a function that measures the performance of a Machine Learning model for given data. Cost Function quantifies the error between predicted values and expected values and presents it in the form of a single real number. Depending on the problem Cost Function can be formed in many different ways. The purpose of Cost Function is to be either:

* Minimized - then returned value is usually called cost, loss or error. The goal is to find the values of model parameters for which Cost Function returns as small a number as possible.
* Maximized - then the value it yields is named a reward. The goal is to find values of model parameters for which the returned number is as large as possible.

The most common cost function is a quadratic function. It is usually an average value since we just need a single cost value for our model.

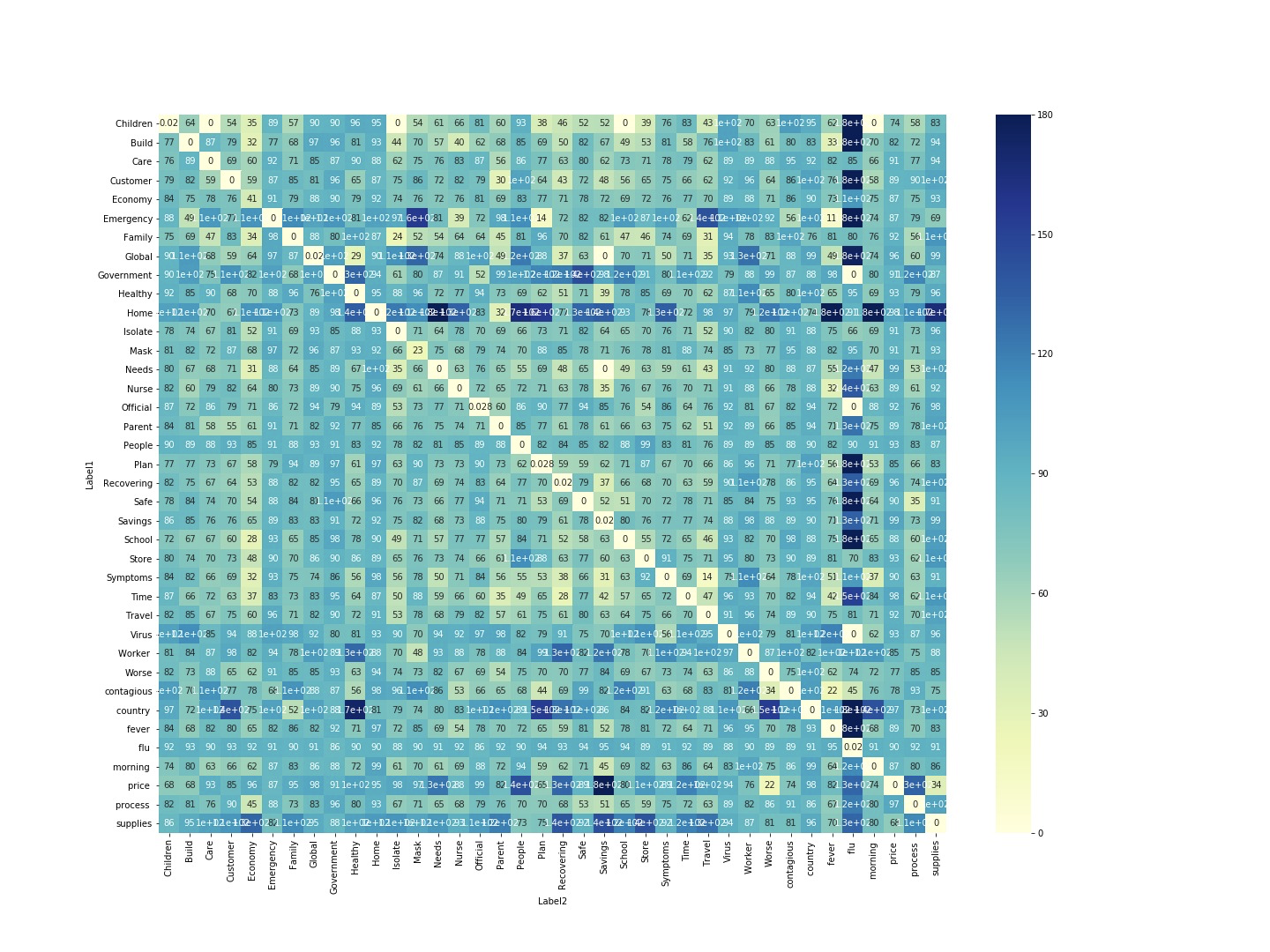
1. **Dot Product**[[12]](#footnote-12) - The set of documents in a collection may be viewed as a set of vectors in a vector space. To compensate for the effect of 2 different document lengths, we choose to use the standard way of quantifying the similarity between two documents d1 and d2 is to compute the cosine similarity of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$

|  |  |
| --- | --- |
| \begin{displaymath} \mbox{sim}(d_1,d_2)= \frac{\vec{V}(d_1)\cdot \vec{V}(d_2)}{\vert\vec{V}(d_1)\vert \vert\vec{V}(d_2)\vert}, \end{displaymath} |  |

where the numerator represents the dot product (also known as the inner product) of the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$, while the denominator is the product of their Euclidean lengths.

1. **Sum of Squares** - Besides simply telling you how much variation there is in a data set, the sum of squares is used to calculate other statistical measures, such as variance, standard error, and standard deviation. These provide important information about how the data is distributed and are used in many statistical tests. To find the accuracy of our model we are checking for a higher sum of square scores value.

## **Heatmap Analysis:**



The words or phrases that are not correlated mean they are completely different topics. But the words or phrases that show a cosine similarity of 0, means they are highly similar.

**Highly correlated words:**

Words such as “virus” and “flu”, “contagious” and “fever”, “emergency” and “fever” are showing high similarity so that they can be correlated to each other. These words are related and can be meaningful in a given context.

**Less correlated words:**

Words such as “travel” and “people”, “contagious” and “home”, “isolate” and “virus” show very less similarity and they cannot be correlated to each other. These words do not go together or they can mean different things.

**Zero Degree Vectors or Vectors that have smaller Angles - Similar Vectors**

In general context, supply has a positive correlation with price. Price and Supplies in heat map have a value of 34 degrees which means that these vectors are closer to similarity angles. When two different phrases or labels such as “price” and “worse” are analyzed in this model, it gives a result of 22 degrees corresponding to a similarity angle.

**90 Degree Vectors or Vectors that have higher Angles - Orthogonal Vectors**

Words such as “Nurse” and “Care” are related in general context but when tested on the model resulted in 79 degrees which means that they are not related and not similar. Other words such as Store and government or Store and Emergency which are unrelated in general context resulted in 90-degree value representing the orthogonality of these phrases.

**180 Degree Vectors or Vectors that have above 90-degree Angles - Opposite Vectors**

“Country” and “Flu”, “Fever” and “flu” have an angle of 180 degrees, which states that they are completely opposite to each other. Also, “Country” and “fever” have an angle of more than 150 degrees. This result can be compared to a mathematical formula such as

“If “a” and “b” are similar,

“b” and “c” are similar then

“a” and “c” should be equal.”

Therefore, based on the COVID articles data set that is trained and the test data (phrases/synonyms list) that is given, the model generates similar vectors, orthogonal vectors and the vectors that are completely opposite and not related to each other.

## **Hyperparameter Tuning Techniques for the Model:**

**Hyperparameter Tuning -** A model **hyperparameter** is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter has to be set before the learning process begins. A parameter is an internal characteristic of the model and its value can be estimated from data.The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. **Hyperparameter optimization or tuning** is the problem of choosing a set of optimal [hyperparameters](https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)) for a learning algorithm. A hyperparameter is a [parameter](https://en.wikipedia.org/wiki/Parameter) whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined [loss function](https://en.wikipedia.org/wiki/Loss_function) on given independent data. The objective function takes a tuple of hyperparameters and returns the associated loss. [Cross-validation](https://en.wikipedia.org/wiki/Cross-validation_(statistics)) is often used to estimate this generalization performance.[[13]](#footnote-13)

## **Grid Search**

Grid Search is a method of performing hyper parameter tuning to determine the optimal values for any model. This process of finding the best parameters is very significant because the performance of the model is dependent on the values produced here.[[14]](#footnote-14) The GridSearchCV library from sklearn makes our life easy by methodically building and evaluating a model with combinations of all the parameters specified in a grid.[[15]](#footnote-15)

**How does it work[[16]](#footnote-16)?**We need to import the GridSearchCV from the sklearn library. This method performs an exhaustive search over a specified parameter value for an estimator. Important members of the method are ‘fit’ and ‘predict’. GridSearchCV implements a ‘fit’ and ‘score’ method.

1. Estimator - estimator object. This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed.
2. Param\_grid - dictionary or list of dictionaries. Dictionaries with parameter names (str) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.
3. Scoring - str, callable, list/tuple or dictionary, default=None
4. N\_jobs -int, default=None - Number of jobs to run in parallel. None means 1 unless in a joblib.parallel\_backend context. -1 means using all processors.
5. Cv - int, cross-validation generator or an iterable, default=None. Determines the cross-validation splitting strategy.

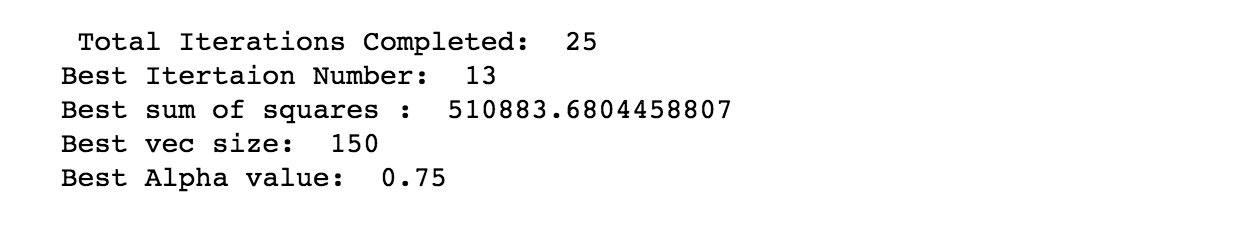
**Challenges in Grid Search approach -** Using Grid Search we found it difficult to include our customized function for the Doc2Vec model where we train the model to find the cosine similarity and degrees between the vectors in any dataset. We couldn’t split the data into test and train as we wanted to find the best parameters for the Doc2Vec function.

The advantage of grid search is that it is guaranteed to find the optimal combination of parameters supplied but there is no guarantee this will be the most perfect solution as it usually finds one from the parameter set supplied to them. This can be overcome with Random Search[[17]](#footnote-17).A major shortcoming of grid search is that when it comes to dimensionality, it will become very difficult if the size of hyperparameters grows exponentially when evaluating them.[[18]](#footnote-18)

**Manual Approach of Grid Search:**

We decided to take a manual approach to find the best parameters for our function. We constructed a nested for loop for evaluating various combinations of the parameters. The model was evaluated, the best score was selected, and the respective parameters were chosen as the best parameters for the model. The parameters we chose to evaluate are the number of iterations, vector size and min alpha.[[19]](#footnote-19)

1. Epochs - Number of iterations (epochs) over the corpus. Defaults to 10 for Doc2Vec.
2. Vector size - Dimensionality of the feature vectors.
3. min\_alpha (float, optional) – Learning rate will linearly drop to min\_alpha as training progresses.

Using the manual approach, each iteration of the nested for loop will be one step in Grid Search.

At the end of the loop, we will get the result with the total number of iterations, best step, best vector size and best alpha value. We were not able to use the GridSearchCV library to perform this hyper parameter tuning since our customized function was not accommodated in the model. We adopted the manual approach to find the best parameters.

## **Random Search**

Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm. In this case, the optimization problem is said to have a low intrinsic dimensionality.[[20]](#footnote-20)

**How does it work?[[21]](#footnote-21)**

Random search is taken from sklearn’s RandomizedSearchCV. We define the hyperparameters to search over before running the search. An important additional parameter to specify here is n\_iter. This specifies the number of combinations to randomly try.

Selecting a low number will decrease our chance of finding the best combination. Selecting a large number will increase our processing time. A few parameters of random search are:

1. Estimator -estimator object. -A object of that type is instantiated for each grid point. This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed.
2. param\_distributions dict or list of dicts - Dictionary with parameters names (str) as keys and distributions or lists of parameters to try. Distributions must provide a rvs method for sampling (such as those from scipy.stats.distributions). If a list is given, it is sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a parameter is sampled using that dict as above.
3. n\_iterint, default=10 Number of parameter settings that are sampled. n\_iter trades off runtime vs quality of the solution.
4. Scoring str, callable, list/tuple or dict, default=None
5. Cv int, cross-validation generator or an iterable, default=None Determines the cross-validation splitting stratege

**Challenges in Random Search:** Using Random Search we found it difficult to include our customized function for the Doc2Vec model where we train the model to find the cosine similarity and degrees between the vectors in any dataset. We couldn’t split the data into test and train as we wanted to find the best parameters for the Doc2Vec function.

Random search is as easy to understand and implement as grid search and in some cases, theoretically more effective. It is performed by evaluating *n* uniformly random points in the hyperparameter space, and selecting the one producing the best performance.   
The drawback of random search is unnecessarily high variance. The method is, after all, entirely random, and uses no intelligence in selecting which points to try. [[22]](#footnote-22)

**Manual Approach of Random Search:**

We decided to take a manual approach to find the best parameters for our function. We constructed a nested for loop for evaluating various combinations of the parameters. The model was evaluated, the best score was selected, and the respective parameters were chosen as the best parameters for the model.We are giving iterations as 5 as list. The parameters we chose to evaluate are the number of iterations, vector size and min alpha.

1. Epochs - Number of iterations (epochs) over the corpus. Defaults to 10 for Doc2Vec.
2. Vector size - Dimensionality of the feature vectors.
3. min\_alpha (float, optional) – Learning rate will linearly drop to min\_alpha as training progresses.

Using the manual approach, each iteration of the nested for loop will be one step in Random Search.

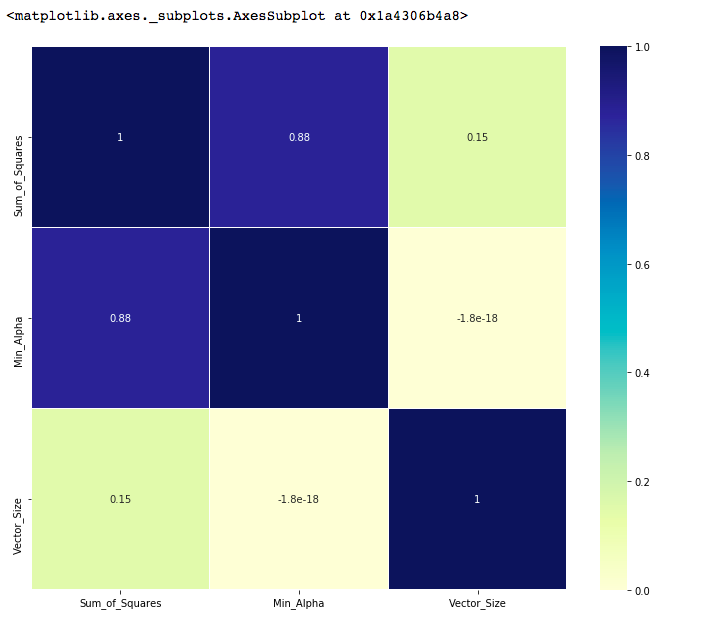


At the end of the loop, we will get the result with the total number of iterations, best step, best vector size and best alpha value. We were not able to use the Random search library to perform this hyper parameter tuning since our customized function was not accommodated in the model. We adopted the manual approach to find the best parameters.

### **Analysis of Hyperparameters And Cost Function Scores**[**¶**](http://localhost:8888/notebooks/Desktop/SCU/Courses/Capstone/Capstone-Fall-Quarter/Sample_Materials/Final_Github_Submissions/Working_Files/Final_Version_NLP_Notebook.ipynb#Step-6:-Analysis-of-Hyperparameters-And-Cost-Function-Scores)

**Analysis of Grid Search Annotated Heatmap of a Correlation Matrix**

A heatmap is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors.[[23]](#footnote-23) A **correlation matrix** is a table showing **correlation** coefficients between variables. Each cell reports a numeric count, like in a standard data table, but the count is accompanied by a color, with larger counts associated with darker colorings.



**Representation of Scores:**

-1: perfect negative linear correlation

+1: perfect positive linear correlation and

0: No correlation

When the data points follow a roughly straight-line trend, the variables are said to have an approximately linear relationship. Denoted by r between -1 and +1

A positive value for r indicates a positive association, and a negative value for r indicates a negative association.

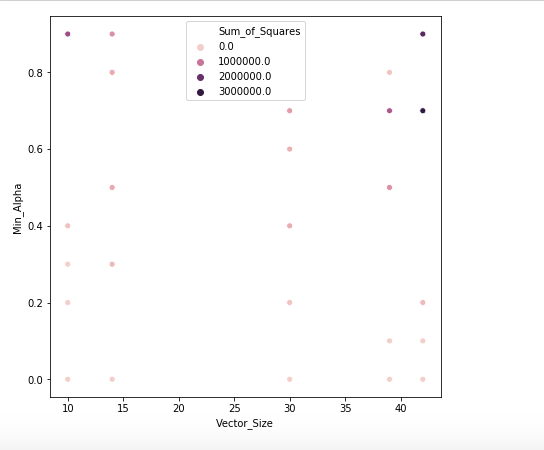
The closer r is to 1 the closer the data points fall to a straight line, thus, the linear association is stronger. The closer r is to 0, making the linear association weaker.

As the Min\_alpha and sum\_of\_squares have the value 0.93 that is closer to 1, denoting the linear association is stronger and tends to have a positive correlation. Whereas vector\_size and sum\_of\_squares has the value 0.15 that is closer to 0, denoting the linear association is weaker and tends to have a negative correlation.

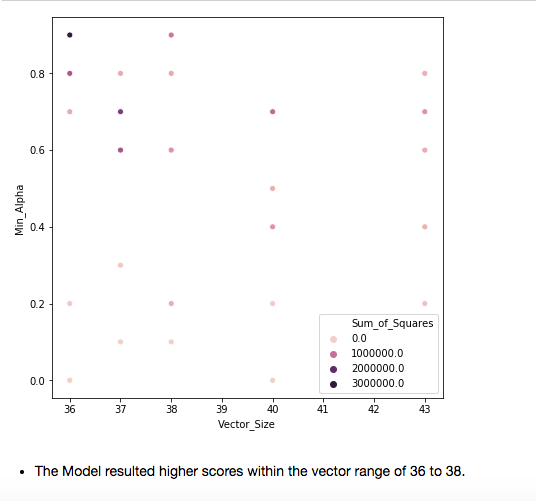
**Random Search Scatterplot Analysis:**

A scatter plot is a diagram where each value in the data set is represented by a dot.The Matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the values of the x-axis, and one for the values of the y-axis.[[24]](#footnote-24)

In the below diagram, it shows the relationship between Vector\_size, Min\_alpha and Sum\_of\_squares. The higher the Vector\_size more the similarity and more the Sum\_of\_squares.



It is also found that the model responded with higher Sum\_of\_squares values with vector\_size ranging from 35-45 and above a 0.6 alpha value.



# **Conclusion:**

We chose a COVID-19 news dataset sourced from various articles, and applied exploratory data analysis and preprocessing to create an NLP pipeline. The tokenized data was converted into vectors using the Doc2Vec model and these vectors were used to find the cosine similarity and orthogonal dimensions between each of them. We observed some irregularities and inconsistencies in the string2vec user-defined function. Therefore, we decided to optimize the model using hyperparameter tuning techniques to achieve an accurate working model. Using Grid Search and Random Search methods we implemented the model tuning. We encountered few challenges while using the inbuilt functions, so we adopted the manual approach for implementing Grid Search and Random Search techniques to tune the model which gave us the accurate working model with the best parameters. We also analyzed the results with the help of seaborn visualization packages to get compelling findings on the model.

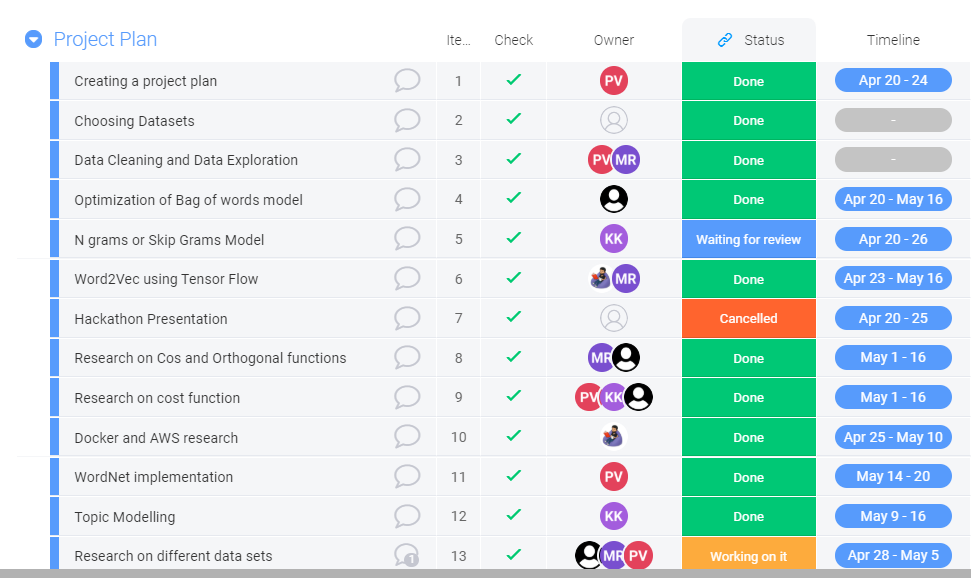
**Appendix:**

# **Success metrics:**

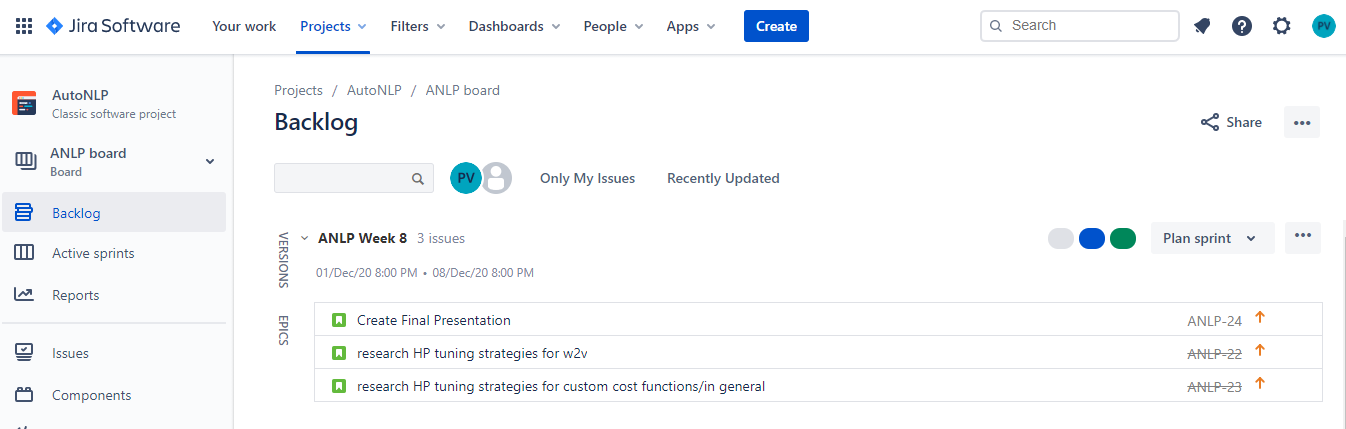
1. Vectorize an arbitrary X input.
2. Performs better than the baseline bag of words model.
3. Using scoring metrics to analyze and check the accuracy of our model.
4. Optimize the model using Hyper Parameter Tuning techniques.
5. Any data set can be used for this model to analyze factors and take measures or decisions.

# **Team Weekly Deliverables**

We created a project plan using monday.com where we could track our task details during the Spring Quarter.

****

Monday.com had a couple of shortcomings so Professor introduced us to using JIRA during the Fall Quarter. We maintain our weekly activities in JIRA allocated to us on every Tuesday by the Professor. We also update our action items, blockers and status for each task as well.

****

# **Team Weekly Meeting Schedule:**

**Spring Quarter**

|  |  |  |
| --- | --- | --- |
| **Days of Meeting** | **Hours of Meeting** | **Platform of Meeting** |
| Wednesday | 1:30 PM to 3:30 PM | Zoom Call |
| Friday | 1:30 PM to 3:00 PM | Zoom Call |
| Saturday | 11:00 AM to 12:00 PM | Zoom Call |

**Fall Quarter**

|  |  |  |
| --- | --- | --- |
| **Days of Meeting** | **Hours of Meeting** | **Platform of Meeting** |
| Tuesday | 7:30 PM to 8:30 PM | Zoom Call |
| Friday | 2:30 PM to 4:00 PM | Zoom Call |
| Monday | 11:00 AM to 12:00 PM | Zoom Call |

# **Overall Challenges faced in the project:**

1. The Labels column in our dataset was not diverse enough for our Doc2Vec model through which we could not identify our outliers.
2. Identifying a good approach for our objective was time consuming because we had to research various methods which would meet our business requirement.
3. Scikit-learn has built in methods such as grid search CV and random search CV that automates the abstract function which computes different iterations and displays the best set of parameters and model. Our customized function using infer vector doesn't work on automated cross validation tuning methods as the set of parameters has to be passed in order to build a model and a new data set in order to evaluate the score.
4. The grid search/random search will instantiate a pipeline of our model for each combination of features, then use cross-validation to score the model and select the best combination of features which was not possible through our data as the splitting of training and test data was not possible on our dataset as it has one text column and the scoring metric is returned on another external data set.
5. Plotting and analyzing heat map correlation matrix was a challenging task.
6. We could not make our model completely automated as we underestimated the time we would need for optimizing it.

# **Deliverables**

As a team, our goal is to provide the following deliverables by the end of the implementation phase as per the timelines mentioned in the project plan:

* Deliver a working NLP model that converts arbitrary input strings into vectors.
* The system architecture design.
* Codebase for the system.
* Documents the process we underwent from idea to implementation.
* A final presentation and demo of our working model to our advisor and faculty members

# **Our Team:**

**DeivaSubhaRanjani Pandurangan Ramamurthy**

DeivaSubhaRanjani is currently pursuing Masters of Science in Information Systems. She has 3 years of work experience in the software industry. She is always enthusiastic to play with data, analyze, derive insights and make valuable business decisions. She has worked as a Product Analyst and played a pivotal role in developing Multi-dimensional Analytics and Reporting Solution (MARS), a self-service BI agile solution enabling business users to create, publish and collaborate on analytics and operational reports. She also has worked with large complex sales datasets to analyze the customer purchase trends and performed competitive market analysis for our client. She has hands-on experience in Java, Python, SQL, VBA and Unix shell scripting. She works well with tools like Tableau, Pentaho, MS excel and MySQL. During her Master’s journey, courses like Data Science with Python, Business Intelligence with Data Warehousing have helped her to take the data, cleanse them and turn them into valuable insights. She is very eager to work with this team and analyze CoronaVirus data and help us understand more about the situation using various Machine Learning Techniques

**Kavitha Kallakere Indrakumar**

Kavitha Kallakere Indrakumar is pursuing Masters in Information Systems.She has worked on different technologies and has 10 years of work experience. She has taken courses on System analysis and design,Database Management system ,Data Science with Python,Big Data and Modelling and Software Project Management.

She wants to work with our team who have experience in different sectors and learn more on Data analyses and build analytics and visualization. She is looking forward to learning more on Machine Learning Techniques and Tools.

**Pooja Varadaraj**

Pooja Varadaraj is a data-driven professional, currently pursuing a Master’s of Science in Information Systems with a focus on Data Analytics at Santa Clara University. Prior to her Masters, she was working as a SAP BI Consultant at Tech Mahindra,India for 3 years. She has solved Business Intelligence issues for the global food giant Nestle by working on live production data. She has hands-on experience in programming languages like python, java and SQL. She has also worked on ETL (pentaho) and data visualizations tools (Tableau and Tableau prep) for her academic course work.

She had taken up courses like data science with python in her master’s which created an interest and passion for her to work on a data science capstone. She is eagerly looking forward to ameliorating her data analytics skills and at the same time contributing to understanding the current pandemic situation arising in the world.

**Mytreyi Reddy**

Mytreyi is a MSIS student from Santa Clara University. She started her career working as a Systems Engineer for 2 and a half years at Tata Consultancy Services. She worked on server-side web applications using C# and Dot Net framework. She has hands-on experience in Java, SQL and she worked on Python, Tableau, MySQL, Pentaho in my academic projects. Currently, she is working more on analytics, accessing data from multiple sources, analyzing and coming up with descriptive insights to make decisions on the business and predictive analytics on the data which would result in better decision making. She is in the phase of learning machine learning models and applying them on the data which would derive accurate results. This learning process has driven her to gain an understanding, prioritizing analytic features, and knowledge on the functionality on the end-end of a product.

1. Quoted by Professor Denis Vrdoljak- <https://www.youtube.com/watch?v=it34KxYMQRQ> [↑](#footnote-ref-1)
2. <http://ilpubs.stanford.edu:8090/917/1/conceptMining-Techrep.pdf> [↑](#footnote-ref-2)
3. <https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72> [↑](#footnote-ref-3)
4. <https://www.datasciencecentral.com/profiles/blogs/top-nlp-algorithms-amp-concepts> [↑](#footnote-ref-4)
5. <http://arxiv.org/pdf/1405.4053v2.pdf> [↑](#footnote-ref-5)
6. <https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/> [↑](#footnote-ref-6)
7. [https://en.wikipedia.org/wiki/Topic\_model#:~:text=In%20machine%20learning%20and%20natural,structures%20in%20a%20text%20body](https://en.wikipedia.org/wiki/Topic_model#:~:text=In%20machine%20learning%20and%20natural,structures%20in%20a%20text%20body.) [↑](#footnote-ref-7)
8. https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/ [↑](#footnote-ref-8)
9. https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/ [↑](#footnote-ref-9)
10. <https://blog.cambridgespark.com/tutorial-build-your-own-embedding-and-use-it-in-a-neural-network-e9cde4a81296> [↑](#footnote-ref-10)
11. <https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/> [↑](#footnote-ref-11)
12. https://nlp.stanford.edu/IR-book/html/htmledition/dot-products-1.html [↑](#footnote-ref-12)
13. https://en.wikipedia.org/wiki/Hyperparameter\_optimization#cite\_note-abs1502.02127-1 [↑](#footnote-ref-13)
14. https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998 [↑](#footnote-ref-14)
15. https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/ [↑](#footnote-ref-15)
16. https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.GridSearchCV.html [↑](#footnote-ref-16)
17. https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624 [↑](#footnote-ref-17)
18. https://www.kdnuggets.com/2020/05/hyperparameter-optimization-machine-learning-models.html [↑](#footnote-ref-18)
19. https://radimrehurek.com/gensim/models/doc2vec.html [↑](#footnote-ref-19)
20. https://paperswithcode.com/method/random-search [↑](#footnote-ref-20)
21. https://paperswithcode.com/method/random-search [↑](#footnote-ref-21)
22. https://docs.google.com/document/d/1F3z\_ozwRRfL7ubhHpyc3pxOBZ2yKw\_as/edit# [↑](#footnote-ref-22)
23. [**https://blog.quantinsti.com/creating-heatmap-using-python-seaborn/#:~:text=A%20heatmap%20is%20a%20two,as%20per%20the%20creator's%20requirement**](https://blog.quantinsti.com/creating-heatmap-using-python-seaborn/#:~:text=A%20heatmap%20is%20a%20two,as%20per%20the%20creator's%20requirement) [↑](#footnote-ref-23)
24. <https://www.w3schools.com/python/python_ml_scatterplot.asp#:~:text=A%20scatter%20plot%20is%20a,%2C12%2C9%2C6%5D> [↑](#footnote-ref-24)