Short Text Mining by Comparative Clustering and Accuracy Measurement

Ayushi Agarwal [1] Simon Cheng Liu [2]

[1] Graduate Student, Carnegie Mellon University, Pittsburgh, PA, U.S.A. This research took place when [1] is Text Mining Intern at Shenji Intelligence Company, Beijing, China. [2]AI Researcher at Shenji Intelligence Company, Beijing, China

ABSTRACT – Majority of organizations that deal with customer related data aim to know the right information from their customers and ways to develop better customer experience. It becomes more essential when the customers are facing issues in installation of a software product or need some assistance. If their issues are well addressed, then the workforce that is spending time in resolving these matters would be better working on something else and the issues will be addressed in the customer FAQs in the company or the product website/manual. This is what our research aims to achieve. It is our goal to automate the clustering of conversations between the customers and the staff or among the customers into most commonly discussed topics and problems of interest and address customer problems before they arise.

There have been researches done related to text mining, text clustering, topic modelling and classification. But none aims at minimizing the human effort, doing short text clustering of chat data over a large corpus or measure the accuracy and minimize runtime effectively.

Our research and analysis overcame all the above-mentioned challenges and we efficient ly developed an algorithm and Python implementation for the same.

I. INTRODUCTION

There has been a lot of research and analysis in the field of text mining [21] and the purpose of this research is to assess the previous development and suggest the best methodologies for efficient short text sentence clustering. Additionally, this research aims to develop an algorithm that effectively clusters the much difficult to understand and assess, large amount of chatroom data and then measure the accuracy of the clustering algorithms by doing a comparative study.

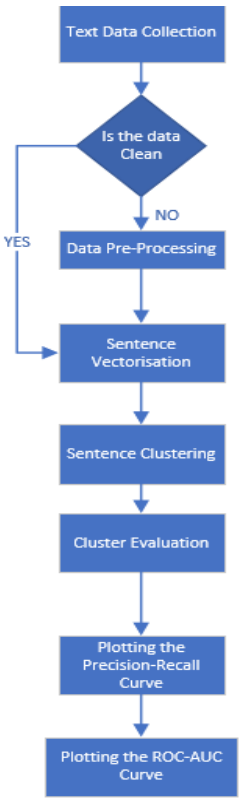


Fig 1: The flowchart below represents process-flow of managing the text data and ways of evaluation in their order.

Process Flow Description: This research deals with the Ubuntu Dialogue Corpus, a dataset that contains almost 1 million multi-turn dialogues, with a total of over 7 million utterances and 100 million words. This provides a unique resource for research into building dialogue managers based on neural language models that can make use of large amounts of unlabelled data. Available at (http://dataset.cs.mcgill.ca/ubuntu-corpus-

1.0/). The dataset has both the multi-turn property of conversations in the Dialog State Tracking Challenge datasets, and the unstructured nature of interactions from microblogging services such as Twitter. **Initially we proceeded with the unlabelled dataset for evaluation and recorded our results**. Then for assessing the accuracy of our results, after studying the dataset carefully, **we labelled the dataset according to the chats that contain windows related conversations (labelled: 1) and that do not have windows related conversations (labelled: -1).** At various instances during the research, we assessed the subsets of the dataset, mainly having 10,000 conversations, 20,000 conversations, 100,000 conversations for the faster evaluation of the algorithms.

The data is very noisy, unlike the text data clustered in previously done researches which is mainly news articles or other kinds of topics of interest which contain no spelling errors and other grammatical mistakes. Chatroom data differs in itself by large amounts as it largely depends on the way people communicate while chatting with someone. It is unique to each individual and they sometimes use different ways to communicate the same message. For obtaining efficient results from clustering mechanisms, pre-processing is another key step and discussed in further section of the research.

Once we pre-process the data, we vectorize the text conversations. The biggest challenge lies into effectively vectorizing the text data available. Various text vectorization techniques developed in the past **like Word2Vec [8], Doc2Vec [9], TF-IDF count vectorizer, NMF and LDA Topic Modelling Mechanism** have been studied and implemented for the process and a comparative result is obtained.

Followed by vectorization, is the clustering process. 9 most efficient clustering algorithms like **K-means, Mean-Shift, Spectral Clustering, Ward Clustering, Agglomerative Clustering, Affinity Propagation, DBSCAN, Gaussian Mixture, Spectral Clustering** have been implemented with each vectorized dataset and the clusters are observed and evaluated.

After assessing the clustering results, we plot the clustering evaluation curves which are two independent random clustering with equal cluster number that measure the **Adjusted Random Score, V-measure score, Mutual Information Score and Adjusted Mutual Information Score.** We also plotted the graph to measure the clustering for random uniform labelling against reference assignment with 2 classes for the same.

Additionally, we verified the accuracy of the clustering results using the **Precision-Recall curve and the ROC-AUC curve** and recorded our observations.

With this project we aim to develop an automated clustering and accuracy checking mechanism to assess the text data and even the most difficult of its kind, the chatroom data using the Python language framework and key features of the libraries like **Gensim [10], Sklearn [11], Numpy, Matplot Library.** We have aimed at using the unsupervised learning to reduce human effort but have referred to some supervised mechanisms for testing purposes. During the process of cluster evaluation and writing the code we have also worked on minimizing the time required to run the algorithms and successfully achieved a satisfactory runtime for each process.

II. LITERATURE SURVEY

As discussed above, our clustering and accuracy measuring mechanism is developed after studying and evaluating numerous research methodologies and techniques

related to short text mining and clustering labelled and unlabelled data.

An informative research accomplished by Tamanna Siddiqui and Parvej Aalam [1] talks about the challenges faced by short text clustering mechanisms and suggests solutions theoretically. It was interesting to bring the methods suggested to bring to life and check the accuracy of the implemented techniques.

Le and Mikolov [2] in their paper introduced the Doc2Vec model which is a derived model or an extension of the Word2Vec technique. Doc2Vec is used to vectorize the sentences into vectors of n-dimension without affecting the sentence order. It has been used in our implementation and the sentence vectors are used for clustering and measuring the accuracy of the clusters. Though our implementation deals with the use of Doc2Vec technique, our experimental results align with the findings of Kim and Cho [3] about the inefficiency of the Doc2Vec and Word2Vec model to pull out impressive clustering results with higher accuracy and also the detailed study of the Bag of Words model which was previously used to vectorize text data. The paper gives a detailed insight to all the methods and implementations and serves as an important part of our research.

One useful research that dealt with the careful analysis of the Doc2Vec model and its effective implementation was by Lau and Baldwin [4] which discusses the effectiveness of the Doc2Vec model when the words are taken without order consideration compared to the model where the word order is taken into account. For short text clustering and vectorization this technique didn't work out effectively even after experimenting with various hyperparameters and measuring accuracy particularly with a hugely noisy and large dataset like ours.

The common drawbacks of majority of the previous the researches is that the (i) Methods and the suggested algorithms are not completely automated, they require a lot of manual work and understanding (ii) The algorithms and the methods suggested are only theoretical or statistical which are convincing on paper but difficult and extremely challenging to bring to an implementation (iii) The comparison is limited, by limited we mean, there is always or most of the times comparison between two to three techniques ignoring the other available methods or possibilities. Our research and implementation deal with four methods to vectorize the text data eight methods to cluster it and then obtaining a perfect match to attain the best results for accuracy and minimizing the human effort in the field of short text clustering and classification for accuracy prediction.

Looking at our research and initial outcomes that led to the further research and successful implementation of the algorithms are discussed as follows. For the Unlabelled chat corpus: We worked with the sample with the 10,000 conversations from the users and removed the labels that kept each conversation together. The only column we had is content.

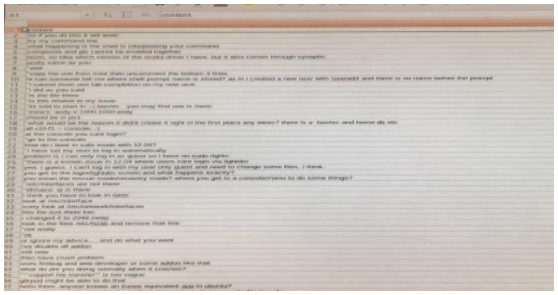


Fig 2: The unlabelled chat dataset in csv format with only one column, content.

After the data was collected, we worked with the pre-processing. The pre-processing steps for the Labelled and unlabelled dataset.

Steps in pre-processing the data (i) Lowercasing the text content. (ii) make a list of stop-words to remove from the data as they add to the noise and do not contribute to the efficient clustering process. (iii) for the unlabelled data, the first column contains the column name which is unnecessary, therefore remove the first line of the dataset. (iv)Create sentence labels and split sentences into words and assign sentence ID.

For example: "**What happening is the shell is interpreting your command**" becomes "**happening shell interpreting command**" .

The next step is text vectorization. For the both labelled and unlabelled data, we used the following vectorizers:

(i) **Doc2Vec:** Doc2Vec [13] is a sentence vectorizer that is a part of the genism library in python and was introduced by Mikilov and Le as discussed above. It is an extension of the Word2Vec model which is used to generate representation vectors out of words. Its representation is created using 2 algorithms continuous Bag-of-Words (CBOW) and the Skip-Gram model. CBOW model works by representing each word as a feature vector and training it. And unlike the CBOW model the skip-gram model words in the opposite way by assigning one vector to a group of words surrounding each other. As an extension of the Word2Vec model, the Doc2Vec model, creates a numeric representation of the document regardless of the length by adding another vector called Paragraph ID as shown in Fig3.

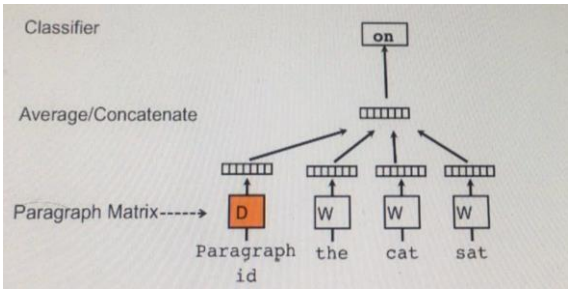


Fig 3: Vector assignment representation [5] through the Doc2Vec model

This is a feature vector and document unique. The hyperparameters of the Doc2Vec model of the genism library can be easily tuned with the key hyper-parameters as:
Vector size= 300 (Vector size was reduced to 2 for testing purposes at many times for 2 dimensional efficient visualisation)
window= 15; min_count= 1; workers=4; alpha=0.1; min_alpha=0.00
For an unlabelled dataset of 10,000 conversations we received 10000 vectors and similarly for labelled conversation of 20,000 sentences we received 20,000 unique vectors and so on.

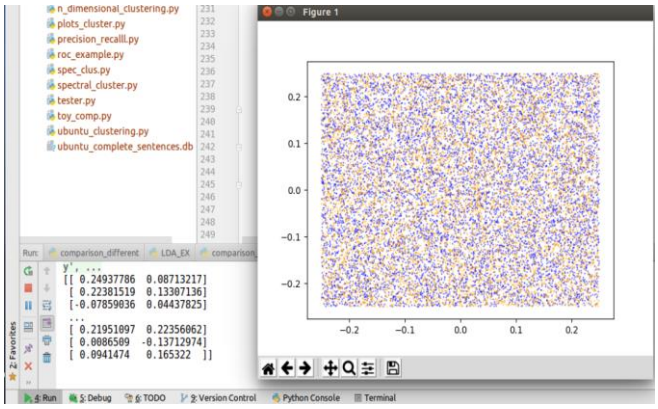
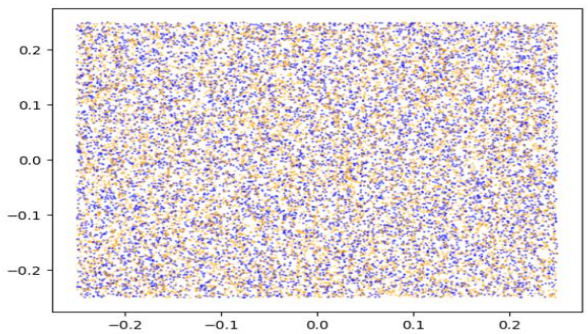


Fig4: Doc2Vec vector representation in 2D space and the vectors obtained for 20,000 labelled sentences



A closer look at the vectorization by Doc2Vec: It is spread across the entire vector space.

(ii) **Count Vectorizer:** Count Vectorizer [14] is another technique used to convert the text data into a matrix of token counts. It is also used in our algorithms and coding process. In python, count vectorizer is a part of the Sci-Kit learn library. It includes a default analyser in its documentation called **WordGramAnalyser** which is responsible to convert the text to lowercase, removal of accents, token extraction and filtering stop words. It creates a sparse matrix which is called a smatrix which has elements stored in a coordinate format.

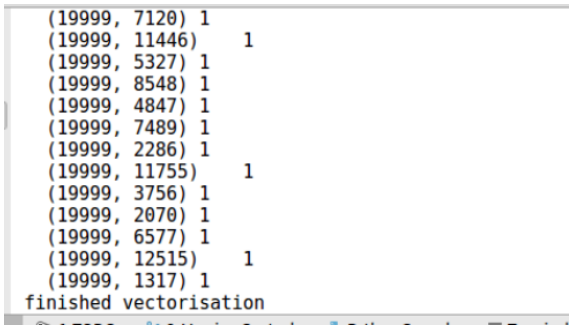


Fig5: Snippet of the vectors Extracted from the Labelled dataset of 20,000 sentences from the chat conversations using count vectorizer

(iii) **TF-IDF vectorizer:** TF-IDF vectorizer [15] is another way of converting the raw text data into a matrix of TF-IDF features. It is a feature of the Sci-Kit learn library in python. TF-IDF eliminates the problem of words being vectorized because of the high frequency and not because of their actual weight or value in the sentence. It uses the logarithmic scale to do that. It normalizes the vector of each word in the sentence so as to give equal weight to frequent and in-frequent words and to remove the bias. It serves an important function in the algorithm and our analysis.

Term Frequency is calculated as:

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

Inverse Document frequency is calculated as:

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Where N is the total number of documents in the corpus. N=|D|

$|\{d \in D : t \in d\}|$ is the number of documents where the term t appears

Therefore, TF-IDF is calculated as:

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

(iv) **NMF and LDA Topic Modelling [16]:** It stands for Non-Negative-Matrix Factorization and Latent Dirichlet Allocation. It works on the text corpus by extracting additive models of the topic structure of the corpus. The output comes as a list of topics and the number of topics are chosen as chosen by the programmer and then the list of the terms is shown irrespective of the weight.

The default parameters are: **n_samples, n_features** and **n_topics/n_components** which in our case is restricted to 2 as we want only 2 topics/clusters our data to be divided in.

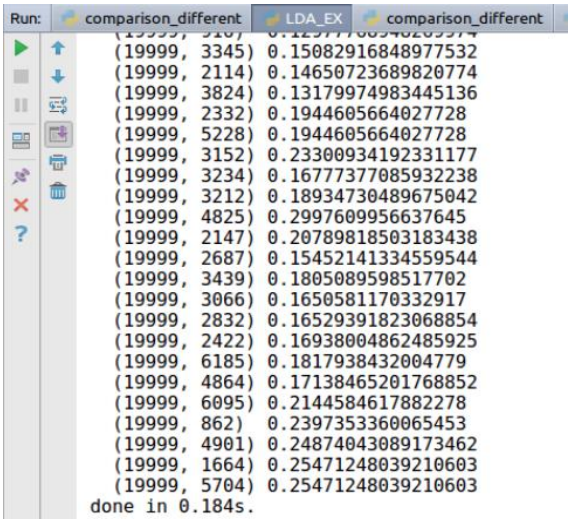


Fig6: Vectors Extracted from the Labelled dataset of 20,000 sentences from the chat conversations using TF-IDF vectorizer

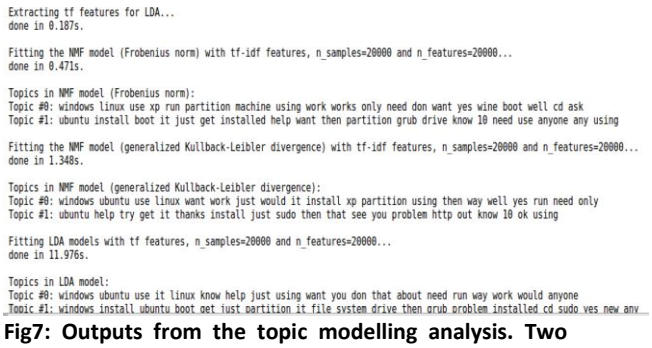


Fig7: Outputs from the topic modelling analysis. Two topics are obtained.

After vectorizing the text data, we need to cluster [12] the obtained text vectors. We have used the Sci-Kit learn library for the clustering algorithms and have done a comparative study on the time taken by each algorithm to cluster our different types of the subsets of the dataset and also about the type of clusters formed. We worked with the following clustering algorithms for both the labelled and the unlabelled dataset for all sizes:

(i) **K-Means:** The basic idea of K Means clustering is to form K seeds first, and then group observations in K clusters on the basis of distance with each of K seeds. The observation will be included in the n^{th} seed/cluster if the distance between the observation and the n^{th} seed is minimum when compared to other seeds. We have used the mini-batch K-means approach for our dataset to optimize the process.

The **Mini-batch K-means** is a variant of the K-means algorithm which uses mini-batches to reduce the computation time, while still attempting to optimize the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution.

The algorithm iterates between two major steps, similar to vanilla k-means. In the first step, b samples are drawn randomly from the dataset, to form a mini-batch. These are then assigned to the nearest centroid. In the second step, the centroids are updated. In contrast to k-means, this is done on a per-sample basis. For each sample in the mini-batch, the assigned centroid is updated by taking the streaming average of the sample and all previous samples assigned to that centroid. This has the effect of decreasing the rate of change for a centroid over time. These steps are performed until convergence or a predetermined number of iterations is reached.

(ii) **Affinity Propagation:** It creates clusters by sending messages between pairs of samples until convergence. A dataset is then described using a small number of exemplars, which are identified as those most representative of other samples. The messages sent between pairs represent the suitability for one sample to be the exemplar of the other, which is updated in response to the values from other pairs. This updating happens iteratively until convergence, at which point the final exemplars are chosen, and hence the final clustering is given.

Affinity Propagation can be interesting as it chooses the number of clusters based on the data provided. For this purpose, the two important parameters are the preference, which controls how many exemplars are used, and the damping factor which damps the responsibility and availability messages to avoid numerical oscillations when updating these messages.

The main drawback of Affinity Propagation is its complexity. The algorithm has a time complexity of the order $O(N^2T)$, where N is the number of samples and T is the number of iterations until convergence. Further, the memory complexity is of the order $O(N^2)$ if a dense similarity matrix is used, but reducible if a sparse similarity matrix is used. This makes Affinity Propagation most appropriate for small to medium sized datasets.

As we will further see, because of the above reason, this is not the best algorithm for clustering on our dataset as it is a large dataset and is therefore further ruled out from the analysis. For two major reasons: (i) Inconsistent and noisy cluster formation (ii) long processing time for the algorithm which makes it really slow.

(iii) **Mean Shift:** It aims to discover blobs in a smooth density of samples. It is a centroid based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate duplicates nearby to form the final set of centroids.

Given a candidate centroid x_i for iteration t, the candidate is updated according to the following equation:

$$x_i^{t+1} = x_i^t + m(x_i^t)$$

Where $N(x_i)$ is the neighborhood of samples within a given distance around x_i and m is the mean shift vector that is computed for each centroid that points towards a region of the maximum increase in the density of points. This is computed using the following equation, effectively updating a centroid to be the mean of the samples within its neighborhood:

$$m(x_i) = \frac{\sum_{x_j \in N(x_i)} K(x_j - x_i) x_j}{\sum_{x_j \in N(x_i)} K(x_j - x_i)}$$

The algorithm automatically sets the number of clusters, instead of relying on a parameter bandwidth, which dictates the size of the region to search through. This parameter can be set manually but can be estimated using the provided estimate_bandwidth function, which is called if the bandwidth is not set.

The algorithm is not highly scalable, as it requires multiple nearest neighbor searches during the execution of the algorithm. The algorithm is guaranteed to converge, however the algorithm will stop iterating when the change in centroids is minimum.

Labelling a new sample is performed by finding the nearest centroid for a given sample. Since the algorithm automatically sets the number of clusters, as stated above, the programmer has no control over the clusters number formed as it will be observed in our case.

(iv) **Spectral Clustering:** It does a low-dimension embedding of the affinity matrix between samples, followed by a K-Means in the low dimensional space. Spectral Clustering requires the number of clusters to be specified. It works well for a small number of clusters but is not advised when using many clusters.

For two clusters, it solves a convex relaxation of the normalized cuts problem on the similarity graph: cutting the graph in two so that the weight of the edges cut is small compared to the weights of the edges inside each cluster. This criterion is especially interesting when working on images: graph vertices are pixels, and edges of the similarity graph are a function of the gradient of the image.

(v) **Ward and Agglomerative Clustering:** Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.

Agglomerative Clustering object performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together. The linkage criteria determine the metric used for the merge strategy:

Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.Maximum or complete
linkage minimizes the maximum distance between observations of pairs of clusters.

Average linkage minimizes the average of the distances between all observations of pairs of clusters.

Agglomerative Clustering can also scale to large number of samples when it is used jointly with a connectivity matrix but is computationally expensive when no connectivity constraints are added between samples: it considers at each step all the possible merges.

(vi) **DBSCAN:** The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples).

More formally, we define a core sample as being a sample in the dataset such that there exist min_samples other samples which are defined as neighbors of the core sample. This tells us that the core sample is in a dense area of the vector space. A cluster is a set of core samples that can be built by recursively taking a core sample, finding all of its neighbors that are core samples, finding all of their neighbors that are core samples, and so on. A cluster also has a set of non-core samples, which are samples that are neighbors of a core sample in the cluster but are not themselves core samples. Intuitively, these samples are on the fringes of a cluster.

Any core sample is part of a cluster, by definition. Any sample that is not a core sample is considered an outlier by the algorithm.

(vii) **Birch:** The Birch builds a tree called the Characteristic Feature Tree (CFT) for the given data. The data is essentially lossy compressed to a set of Characteristic Feature nodes (CF Nodes). The CF Nodes have a number of subclusters called Characteristic Feature subclusters (CF Subclusters) and these CF Subclusters located in the non-terminal CF Nodes can have CF Nodes as children.

The CF Subclusters hold the necessary information for clustering which prevents the need to hold the entire input data in memory. This information includes: Number of samples in a subcluster. Linear Sum - A n-dimensional vector holding the sum of all samples. Squared Sum - Sum of the squared L2 norm of all samples. Centroids - To avoid recalculation linear sum / n_samples. Squared norm of the centroids.

The Birch algorithm has two parameters, the threshold and the branching factor. The branching factor limits the number of subclusters in a node and the threshold limits the distance between the entering sample and the existing subclusters.

This algorithm can be viewed as an instance or data reduction method, since it reduces the input data to a set of subclusters which are obtained directly from the leaves of the CFT. This reduced data can be further processed by feeding it into a global clusterer. This global clusterer can be set by n_clusters. If n_clusters is set to None, the subclusters from the leaves are directly read off, otherwise a global clustering step labels these subclusters into global clusters (labels) and the samples are mapped to the global label of the nearest subcluster.

Algorithm description:

- A new sample is inserted into the root of the CF Tree which is a CF Node. It is then merged with the subcluster of the root, that has the smallest radius after merging, constrained by the threshold and branching factor conditions. If the subcluster has any child node, then this is done repeatedly till it reaches a leaf. After finding the nearest subcluster in the leaf, the properties of this subcluster and the parent subclusters are recursively updated.

- If the radius of the subcluster obtained by merging the new sample and the nearest subcluster is greater than the square of the threshold and if the number of subclusters is greater than the branching factor, then a space is temporarily allocated to this new sample. The two farthest subclusters are taken and the subclusters are divided into two groups on the basis of the distance between these subclusters.

- If this split node has a parent subcluster and there is room for a new subcluster, then the parent is split into two. If there is no room, then this node is again split into two and the process is continued recursively, till it reaches the root.

(viii) **Gaussian Mixture:** A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

Sci-kit-learn implements different classes to estimate Gaussian mixture models, that correspond to different estimation strategies, detailed below.

The GaussianMixture object implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models, and compute the Bayesian Information Criterion to assess the number of clusters in the data.
A GaussianMixture.fit method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belongs to using the GaussianMixture.predict method.

The GaussianMixture comes with different options to constrain the covariance of the difference classes estimated: spherical, diagonal, tied or full covariance.

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large `n samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large `n samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

Fig8: Comparative Study among [6] different clustering algorithms used in the clustering process

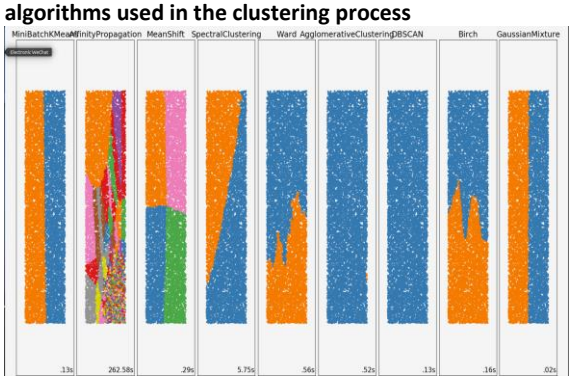


Fig9: For the unlabelled 10,000 sentence dataset, we observe the above clusters for each algorithm.
We have restricted the n_ cluster to be 2. Since, Mean Shift algorithm automatically chooses its clusters, it gives 3 clusters. (Doc2Vec vectorizer used)

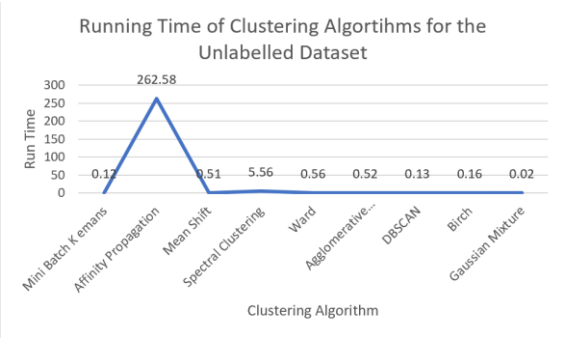


Fig10: We observe that, Affinity propagation algorithm takes 262.58 seconds to run which is the longest running time and gives irrelevant clusters. We choose to not assess it in our further analysis.

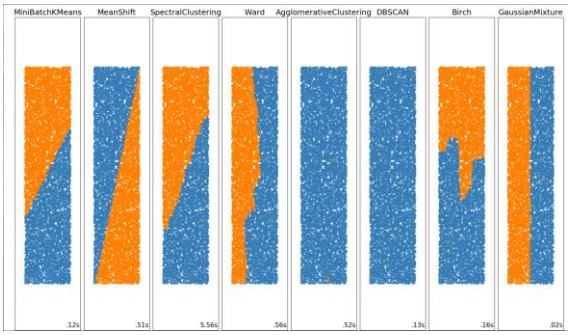


Fig11: Clusters for the Unlabelled dataset for different algorithms after removal of the Affinity Propagation Algorithm, n_cluster=2. (Doc2Vec vectorizer used)

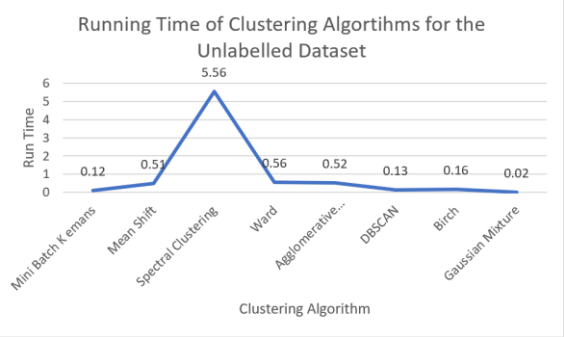


Fig12: Now we observe that the run time of the entire process is significantly reduced and Spectral Clustering takes maximum run time as compared to the other algorithms.

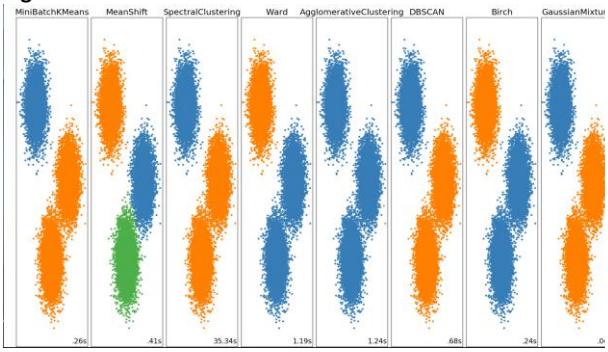


Fig13: For the labelled dataset with 20,000 sentences, we observe the above clusters for each algorithm. Here n_clusters=2. Since, Mean Shift algorithm automatically chooses its clusters, it gives 3 clusters. (Doc2Vec vectorizer used)

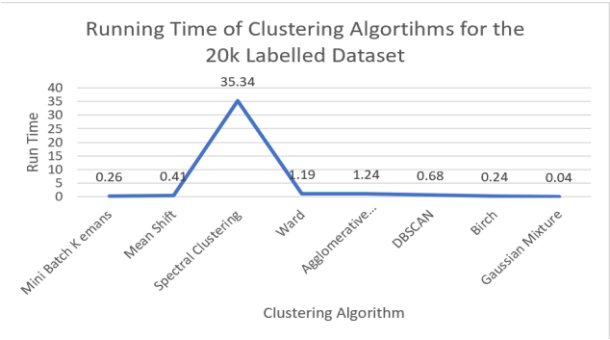


Fig14: We observe the above runtimes and Spectral Clustering takes maximum run time as compared to the other algorithms.

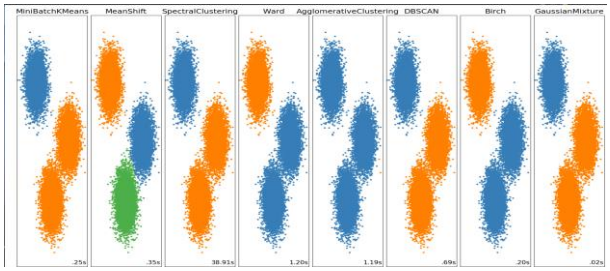


Fig15: For the labelled dataset with 20,000 sentences, we observe the above clusters for each algorithm. Here n_clusters=2. Since, Mean Shift algorithm automatically chooses its clusters, it gives 3 clusters. (Count vectorizer vectorizer used) Results can be observed to be similar to Doc2Vec results.

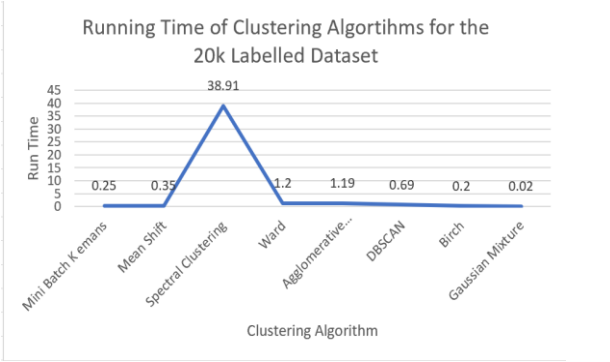


Fig16: We observe the above runtimes and Spectral Clustering takes maximum run time as compared to the other algorithms. For the count vectorizer vectorization as-well.

Cluster Performance Evaluation: Moving on from the sentence clustering and after recording our observations we evaluate the clustering accuracy using various evaluation techniques. Evaluating the performance of a clustering algorithm is not as trivial as counting the number of errors or the precision and recall of a supervised classification algorithm. In particular any evaluation metric should not take the absolute values of the cluster labels into account but rather if this clustering define separations of the data similar to some ground

truth set of classes or satisfying some assumption such that members belong to the same class are more similar that members of different classes according to some similarity metric. All are calculated with the help of the Sci-Kit Learn library.

(i) **Adjusted Rand Score**: Given the knowledge of the ground truth class assignments labels_true and our clustering algorithm assignments of the same samples labels_pred, the adjusted Rand index is a function that measures the similarity of the two assignments, ignoring permutations and with chance normalization.

It can be explained with the following example [7]:

```
>>> from sklearn import metrics
>>> labels_true = [0, 0, 0, 1, 1, 1]
>>> labels_pred = [0, 0, 1, 1, 2, 2]

>>> metrics.adjusted_rand_score(labels_true, labels_pred)
0.24...
```

Fig17: Gives us an example of the Adjusted Random Score where we can see that the score represents the relation between the predicted and the true values.

(ii) **Mutual Information based scores:** Given the knowledge of the ground truth class assignments labels_true and our clustering algorithm assignments of the same samples labels_pred, the Mutual Information is a function that measures the agreement of the two assignments, ignoring permutations.

The mutual information of two random variables is defined by:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right),$$

where $p(x,y)$ is the joint probability function of X and Y, and P(x) and p(y) are the marginal probability distribution functions of X and Y respectively.

Two different normalized versions of this measure are available, Normalized Mutual Information(NMI) and Adjusted Mutual Information(AMI). NMI is often used in the literature while AMI was proposed more recently and is normalized against chance.

```
>>> from sklearn import metrics
>>> labels_true = [0, 0, 0, 1, 1, 1]
>>> labels_pred = [0, 0, 1, 1, 2, 2]

>>> metrics.adjusted_mutual_info_score(labels_true, labels_pred)
0.22504...
```

One can permute 0 and 1 in the predicted labels, rename 2 to 3 and get the same score:

```
>>> labels_pred = [1, 1, 0, 0, 3, 3]
>>> metrics.adjusted_mutual_info_score(labels_true, labels_pred)
0.22504...
```

Fig18: Gives us an example of the Mutual Information Score [6] where we can see how the score represents the relation between the predicted and the true values.

(iii) **V-measure Score:** Given the knowledge of the ground truth class assignments of the samples, it is possible to define some intuitive metric using conditional entropy analysis.

In particular research by Rosenberg and Hirschberg [22] define the following two desirable objectives for any cluster assignment:

- homogeneity: each cluster contains only members of a single class.
- completeness: all members of a given class are assigned to the same cluster.

Their harmonic mean called **V-measure** is computed by v_measure score.

```
>>> metrics.v_measure_score(labels_true, labels_pred)
0.51...
```

Fig19: Gives us an example [6] of the V-measure Score where we can see how the score represents the relation between the predicted and the true values.

For the Unlabelled Dataset using Doc2Vec Vectorizer:

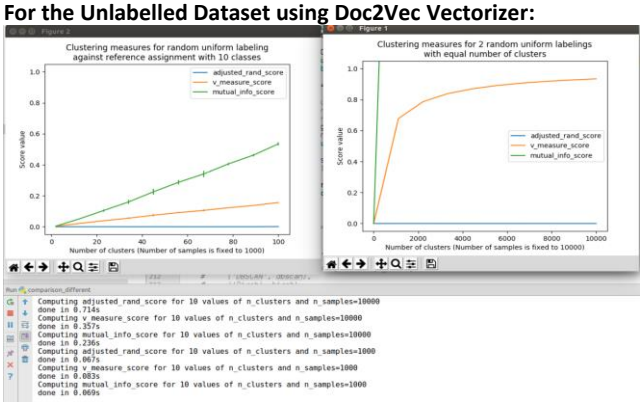


Fig19: Performance evaluation curves for unlabelled dataset with doc2vec vectorization (10,000 Sentences)

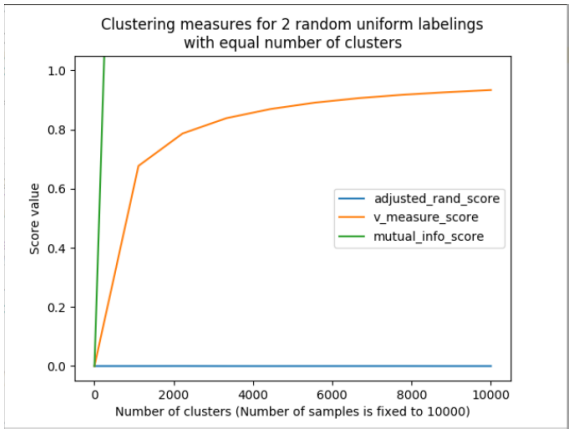


Fig20: Clustering measures for 2 random uniform labelling with equal number of clusters for the unlabelled dataset with doc2vec vectorizer

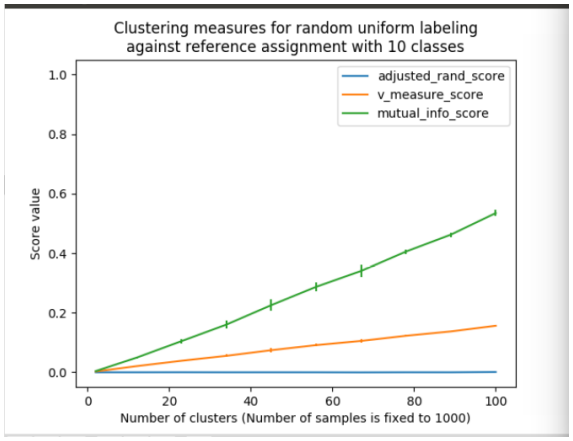


Fig21: Clustering measures for uniform labelling against reference assignment with 10 classes for the unlabelled dataset with doc2vec vectorizer

For the Labelled Dataset using Doc2Vec Vectorizer:

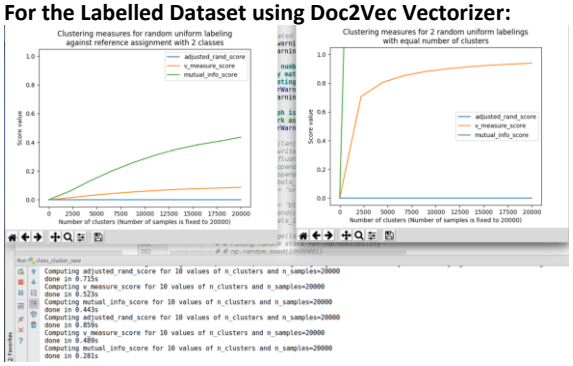


Fig21: Performance evaluation curves for labelled dataset with doc2vec vectorization (20,000 Sentences)

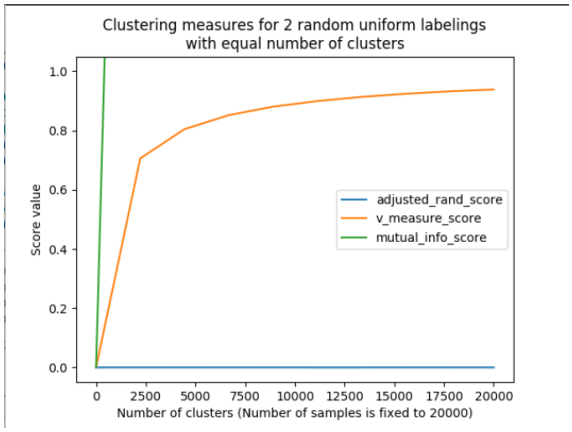


Fig22: Clustering measures for 2 random uniform labelling with equal number of clusters for the labelled dataset with doc2vec vectorizer

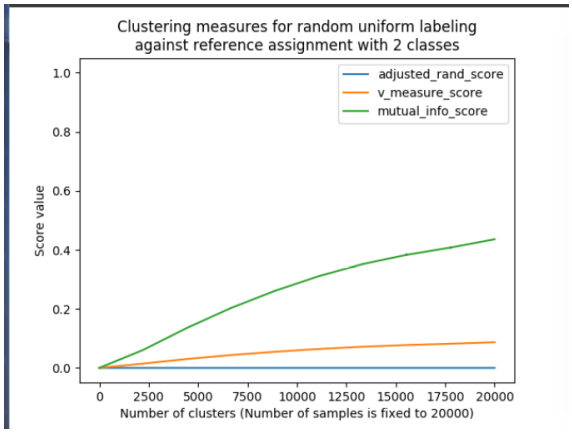


Fig23: Clustering measures for uniform labelling against reference assignment with 10 classes for the labelled dataset with doc2vec vectorizer

We obtain same results as the results obtained from Doc2Vec for the Count Vectorizer cluster evaluation.

Precision Recall [18]: For our dataset, it is essential that we plot and evaluate the output through the precision and recall curve as the data is highly unbalanced and too noisy, therefore we need a useful measure of success prediction.

The precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.

Precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p).

$P = \frac{T_p}{T_p+F_p}$

Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n).

$R = \frac{T_p}{T_p+F_n}$

These quantities are also related to the (F_1) score, which is defined as the harmonic mean of precision and recall.

$F1 = 2\frac{P \times R}{P+R}$

Note that the precision may not decrease with recall. The definition of precision ($\frac{T_p}{T_p+F_p}$) shows that lowering the threshold of a classifier may increase the denominator, by increasing the number of results returned. If the threshold was previously set too high, the new results may all be true positives, which will increase precision. If the previous threshold was about right or too low, further lowering the threshold will introduce false positives, decreasing precision.

Recall is defined as $\frac{T_p}{T_p+F_n}$, where $T_p + F_n$ does not depend on the classifier threshold. This means that lowering the classifier threshold may increase recall, by increasing the number of true positive results. It is also possible that lowering the threshold may leave recall unchanged, while the precision fluctuates.

The relationship between recall and precision can be observed in the stairstep area of the plot - at the edges of these steps a small change in the threshold considerably reduces precision, with only a minor gain in recall.

Average precision (AP) summarizes such a plot as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight:

$\mathrm{AP} = \sum_n (R_n - R_{n-1}) P_n$

where P_n and R_n are the precision and recall at the nth threshold. A pair (R_k, P_k) is referred to as an *operating point*.

AP and the trapezoidal area under the operating points are common ways to summarize a precision-recall curve that lead to different results.

Precision-recall curves are typically used in binary classification to study the output of a classifier. In order to extend the precision-recall curve and average precision to multi-class or multi-label classification, it is necessary to binarize the output. One curve can be drawn per label, but one can also draw a precision-recall curve by considering each element of the label indicator matrix as a binary prediction (micro-averaging).

We have plotted the Binary Classification precision and recall curve as our data is labelled into two classes, windows and non-windows. Since, we need the True Positive Rate, False Negative Rate and False Positive Rate (elements of the confusion matrix) which are only a result of classification, therefore we cannot plot a precision and recall curve without using a classifier. We used the SVM (Support Vector Machines) which are a set of supervised learning mechanisms used for classification and are highly effective in high dimensional spaces for calculating precision recall curve. It's extension LinearSVC Classifier from the Sci-Kit learn library is mainly used to classify the dataset. It is also called Linear Support Vector Classification. Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

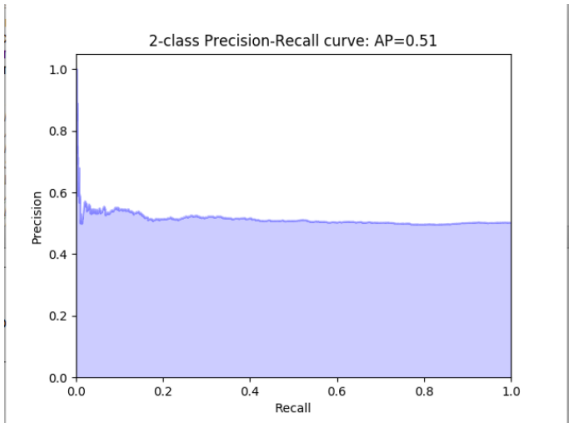


Fig24: The area under the precision recall curve shows that the accuracy is very low 51% to correctly predict the class of the sentences.

For Labelled Dataset with 20,000 sentences using Doc2Vec:

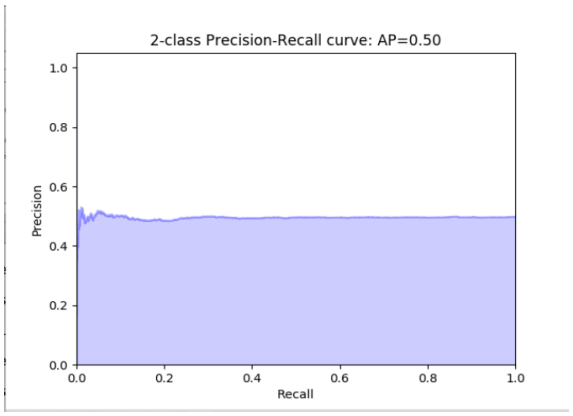


Fig25: The area under the precision recall curve shows that the accuracy is very low 51% to correctly predict the class of the sentences. We have not received better accuracy even after labelling the dataset.

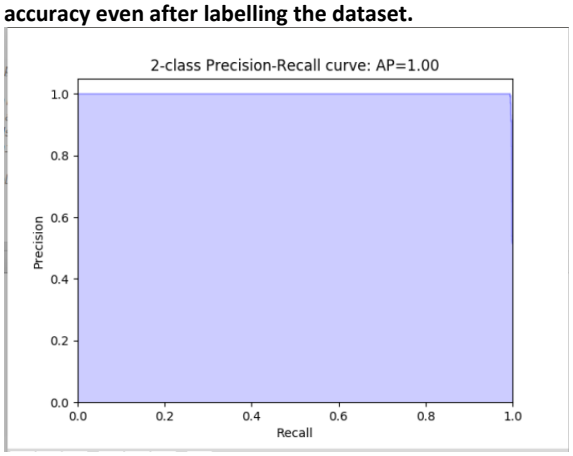


Fig26: The area under the precision recall curve shows that the accuracy is 100% which does not coincide with the clustering results by the Count vectorizer where it was clustering 50% of the windows related sentences in non-windows cluster and vice-versa, therefore this result is also not reliable.

ROC Curve: Example of Receiver Operating Characteristic (ROC) [17] metric to evaluate classifier output quality.

ROC curves typically feature true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the ideal point - a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better.

The steepness of ROC curves is also important, since it is ideal to maximize the true positive rate while minimizing the false positive rate.

We have chosen to plot the ROC with cross validation.

We will observe that our ROC curve results will coincide with the results of the Precision Recall curve.

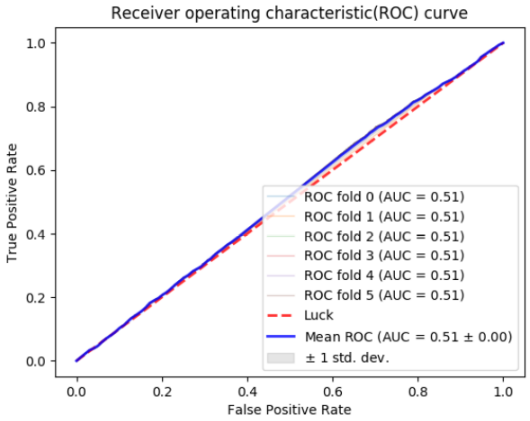


Fig27: ROC Curve for Unlabeled Dataset with Doc2Vec vectorization. The result is unsatisfactory as the pull out rate of true positives is very low.

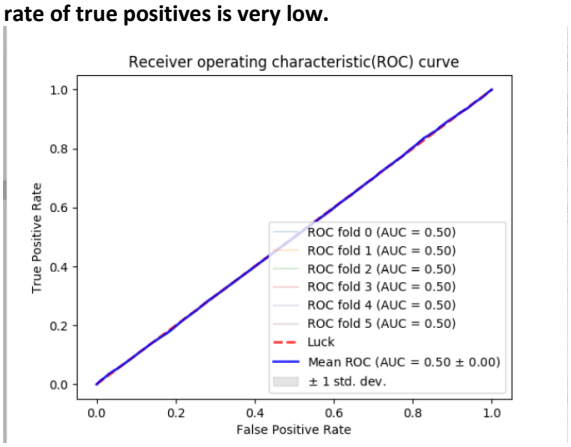


Fig28: ROC Curve for Labelled Dataset with Doc2Vec vectorization. The result is unsatisfactory as the pull out rate of true positives is very low.

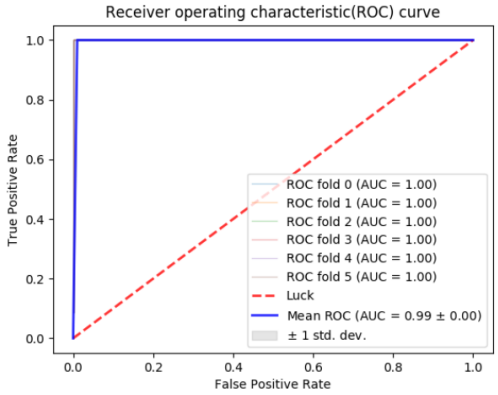


Fig29: ROC Curve for Labelled Dataset with Count vectorization. The result is unsatisfactory as the pull out rate of true positives is very low.

III. THE PROPOSED MECHANISM WITH NMF VECTORIZER

As observed above, we do not get the desired results from the methods used above and after investigation with the code and the algorithms for clustering and analysing the precision-recall curve and the ROC-AUC curve, our understanding comes to the conclusion that the vectorizer for converting the text data to vector format needed to be analysed further and other techniques needed to be used.

We then worked with the NMF method [19] with the TF-IDF Vectorizer and the results were surprisingly better than the above conventional methods used for text clustering. Along with the NMF and TF-IDF vectorization, we also used LDA topic modelling but did not receive satisfactory results as it could not cluster the windows and non-windows related topics separately efficiently.

Topics in LDA model:

Topic #0: windows ubuntu use it linux know help just using want you don that about need run way work would anyone

Topic #1: windows install ubuntu boot get just partition it file system drive then grub problem installed cd sudo yes new any

As we can see that in the above output snippet that the two clusters with 20 most common words in clusters for LDA have windows as the top word therefore, it is not the right approach to choose.

After ruling out the LDA topic modelling of the process, we will see how NMF works and how it helped us reach our goal of effectively clustering our dataset.

NMF working Mechanism: To conduct NMF we formalize an objective function and iteratively optimize it. NMF is an NP-hard problem in general, so we will aim for a good local minimum.

NMF (Nonnegative Matrix Factorization) is a matrix factorization method where we constrain the matrices to be non-negative. In order to understand NMF, we should clarify the underlying intuition between matrix factorization.

Suppose we factorize a matrix X into two matrices W and H so that $X \approx WH$.

Let's look at the objective function.

Although there are some variants, a generally used measures of distance is the frobenius norm (the sum of element-wise squared errors). Formalizing this, we obtain the following objective:

$$minimize \|X - WH\|_F^2 \, w.r.t. \, W, H \, s.t. \, W, H \geq 0$$

Next, we will introduce a couple of optimization methods.

A commonly used method of optimization is the multiplicative update method. In this method, W and H are each updated iteratively according to the following rule:

$$H \leftarrow H \odot \frac{W^T X}{W^T W H}$$

$$W \leftarrow W \odot \frac{X H^T}{W H H^T}$$

where the matrix not being updated is kept constant.

The objective functions can be proved to be non-increasing under this rule. A less formal, but more intuitive explanation of this method

is to interpret this as rescaled gradient descent. Rescaled gradient descent can be written as follows:

$$H \leftarrow H - \eta \odot [W^T W H - W^T X]$$

$$W \leftarrow W - \zeta \odot [W H H^T - X H^T]$$

If we set η to be $\frac{H}{W^T W H}$ and ζ to be $\frac{W}{W H H^T}$, then we obtain the multiplicative update rule.

How is NMF effective in Topic Modelling: NMF is one of the most effective machine learning techniques for topic modelling. Let's imagine that we wanted to decompose a term-document matrix, where each column represented a document, and each element in the document represented the weight of a certain word.

What happens when we decompose this into two matrices? Our dataset comes from a chatroom conversation. The word "windows" would be likely to appear in windows OS related conversations, and therefore co-occur with words like "boot" and "install", here we are not referring to the weights of the respective words in the sentence. Therefore, these words would probably be grouped together into a "windows related" component vector, and each sentence would have a certain n weight of the "windows related" topic. The same stands true for the non-windows or specifically saying "Ubuntu" operating system.

Therefore, an NMF decomposition of the term-document matrix would yield components that could be considered "topics" and decompose each document into a weighted sum of topics. This is called topic modeling and is an important application of NMF.

Another interesting property of NMF is that it naturally produces sparse representations. This makes sense in the case of topic modeling: documents generally do not contain a large number of topics and therefore we are able to differentiate the topics better.

Our Approach with the NMF algorithm:

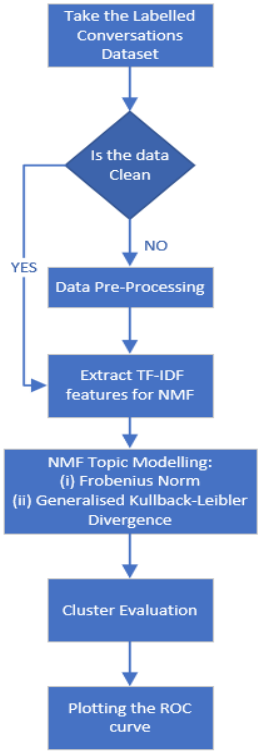


Fig30: The above flowchart represents our workflow with the NMF algorithm and obtaining a Precision-Recall curve in the end.

Dataset: Labelled Sentences Conversation Dataset. As discussed in section I of the paper, we divided our conversations into namely two sections, windows related conversations and non-windows related conversations.
We took a sample of the same dataset which we have referred as the labelled dataset throughout the research paper and took 20k sentences from the sample initially. After that the testing is done on the dataset of 100k conversations to obtain the best results and also give us the idea of the success of the algorithm on larger datasets.

Data Pre-Processing: Data Pre- Processing for the labelled dataset is same as discussed in section II in the preprocessing steps and stop word removal. To maintain uniformity in the process and for efficient comparison we have not preprocessed the data further than the steps discussed in the section II.

Feature Extraction: We have used the TF-IDF vectorizer for extracting features out of the conversations. The parameters are as follows:

n_samples: 20000
n_features: 20000
n_components: 2
n_top_words=20

Here,n_samples are the number of conversations, features are the number of features we desire out of the conversations.
n_components refer to the number of topics we want our text to be clustered into.
n_top_words refer to the number of words with maximum weight we want our output clusters to generate.

NMF Topic Modelling: Let us begin by looking at our output from the NMF topic modelling and then working on it backwards:

Fitting the NMF model (Frobenius norm) with tf-idf features, n_samples=20000 and n_features=20000...
done in 1.271s.

Topics in NMF model (Frobenius norm):
Topic #0: windows linux use xp run partition machine using work works only need don want yes wine boot well cd ask

Topic #1: ubuntu install boot it just get installed help want then partition grub drive know 10 need use anyone any using

Fitting the NMF model (generalized Kullback-Leibler divergence) with tf-idf features, n_samples=20000 and n_features=20000...
done in 3.747s.

Topics in NMF model (generalized Kullback-Leibler divergence):
Topic #0: windows ubuntu use linux want work just would it install xp partition using then way well yes run need only

Topic #1: ubuntu help try get it thanks install just sudo then that see you problem http out know 10 ok using

There are two ways how topics are modelled using the NMF[20]: One is the Frobenius form and the other is the Kullback-leibier Divergence and we received good clusters (separated the windows and non-windows effectively).
In our further analysis we have used the Frobenius form only.
As seen in the above output snippet, we achieved the desired goal of getting windows clustered in one cluster and other cluster does not have windows in the top weighted 20 words. The NMF algorithm works by assigning weight to the words in each sentence and then the word represents the sentence.

For further processing, we take in account a sample of 100k sentences for the analysis as we wish to obtain a more generalized view for the dataset and test our results on a larger corpus.

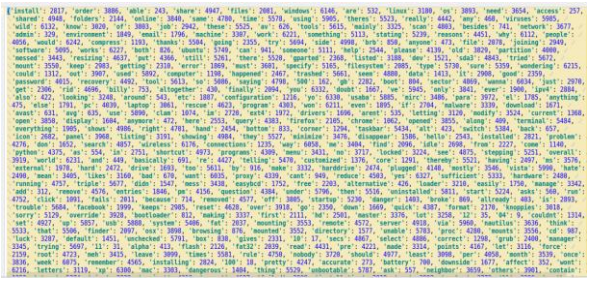


Fig31: A snippet of the output representing the weights assigned to each word by the NMF algorithm.

Extracting the sentence out of the words was another issue we dealt with by matching the word with the topic and added a score to each word by giving it labels as windows related or non-windows related. A word which has higher weight on the windows section is likely to be in the windows topic and vice-versa.

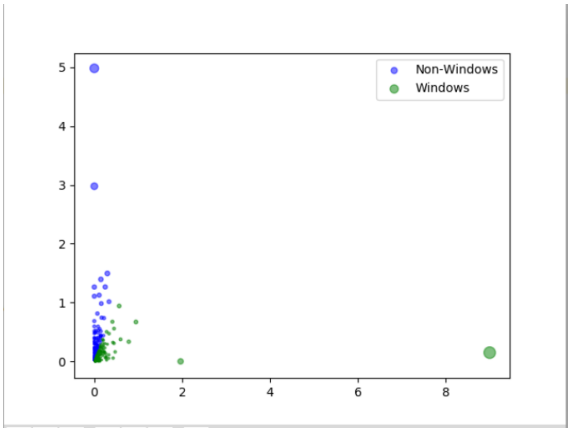


Fig32: Based on the weight the words are put in the graph in their respective places. We can see that the blue dots represent the non-windows clusters and the green represent the words with windows.

We obtained clear clusters according to the weight of the words unlike the results obtained from previous algorithms where there were clusters which had almost 50% of the windows and non-windows data.
After assigning the topic, we compare it to the actual label and then run a comparative test.

Plotting the ROC curve: We extracted the Sensitivity threshold for the dataset and the false positive and true positive rate for each document and then plotted in the ROC curve as discussed below.

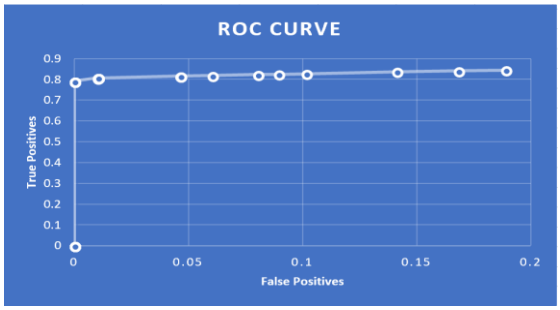


Fig33: Receiver Operating Characteristic Curve is the plot of True Positives vs False Positives and here we have plotted this after comparing our results of the comparison test with the actual labels.

As we can see in the above ROC curve, our results coincide with that of the clustering results and the ROC curve successfully depicts the True-Positive pull-out ratio of more than 80% which is efficient and satisfactory.

The runtime of the algorithm is also minimized to 1-2 seconds which is highly efficient for a large dataset with 100k sentences. We have the following specifications used for the research are CPU- 8 core and 3.6 GHz with i7 processor and 16 GB RAM.

IV. CONCLUSION

As observed in the section II and section III, we are successfully able to achieve our goal of clustering the chat conversations as windows and non-windows.

Most successful method of vectorizing the text conversations is TF-IDF vectorizer and the most successful method of clustering is the NMF topic modelling method.

As said in above sections, for better accuracy and understanding purposes, we had to create windows(1) and non-windows(-1) as two labels. For future challenges and studies, researchers should focus on clustering the unlabelled text data and find ways to measure accuracy without creating labels and achieve the same or better accuracy than achieved in our research using the ROC-AUC curve or Precision-Recall.

Above all, I would like to thank my mentor, Simon Cheng Liu and Dr. Chris Guo at Shenji International Company, Beijing, China to give me an opportunity during my internship to conduct this interesting research and providing continuous support and encouragement throughout the process.

V. REFERENCES

[1]. Ijmcr.in. (2018). [online] Available at: http://ijmcr.in/v3-i6/4%20ijmcr.pdf

[2]. Arxiv.org. (2018). [online] Available at: https://arxiv.org/pdf/1405.4053v2.pdf

[3]. Dm.snu.ac.kr. (2018). [online] Available at: http://dm.snu.ac.kr/static/docs/TR/SNUDM-TR-2015-05.pdf [Accessed 9 Jul. 2018].

[4]. Arxiv.org. (2018). [online] Available at: https://arxiv.org/pdf/1607.05368.pdf

[5]. Anon, (2018). [online] Available at: http://medium.com/scaleabout/a-genle-introduction-to-doc2vec-db3e8cce5e

[6]. Scikit-learn.org. (2018). *2.3. Clustering — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/clustering.

[7]. Scikit-learn.org. (2018). *2.3. Clustering — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/clustering.html#adjusted-rand-index

[8]. Anon, (2018). [online] Available at: http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/

[9]. Kaggle.com. (2018). *Bag of Words Meets Bags of Popcorn | Kaggle*. [online] Available at: https://www.kaggle.com/c/word2vec-nlp-tutorial [Accessed 9 Jul. 2018].

[10]. Radimrehurek.com. (2018). *gensim: topic modelling for humans*. [online] Available at: https://radimrehurek.com/gensim/tutorial.html

[11]. Scikit-learn.org. (2018). *scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/ [Accessed 9 Jul. 2018].

[12]. Brandonrose.org. (2018). *Document Clustering with Python*. [online] Available at: http://brandonrose.org/clustering [Accessed 9 Jul. 2018].

[13]. Towards Data Science. (2018). *Automatic Topic Clustering Using Doc2Vec – Towards Data Science*. [online] Available at: https://towardsdatascience.com/automatic-topic-clustering-using-doc2vec-e1cea88449c

[14]. Scikit-learn.org. (2018).*sklearn.feature_extraction.text.CountVectorizer — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[15]. Scikit-learn.org. (2018). *sklearn.feature_extraction.text.TfidfVectorizer — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html [Accessed 9 Jul. 2018].

[16]. Scikit-learn.org. (2018). *sklearn.decomposition.LatentDirichletAllocation — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html#sklearn.decomposition.LatentDirichletAllocation [Accessed 9 Jul. 2018].

[17].http://scikitlearn.org/stable/auto_examples/model_selection/plot_roc_crossval.html#sphx-glr-auto-examples-model-selection-plot-roc-crossval-py

[18]. Scikit-learn.org. (2018). *Precision-Recall — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

[19]. Scikit-learn.org. (2018). *sklearn.decomposition.NMF — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html

[20]. Scikit-learn.org. (2018). *Topic extraction with Non-negative Matrix Factorization and Latent Dirichlet Allocation — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikitlearn.org/stable/auto_examples/applications/plot_topics_extraction_with_nmf_lda.html

[21].Expertsystem.com. (2018). *Text mining research papers - Expert System*. [online] Available at: https://www.expertsystem.com/text-mining-research-papers/

[22].Aclweb.org. (2018). [online] Available at: http://www.aclweb.org/anthology/D07-1043