UNIVERSITÄT KOBLENZ · LANDAU
Faculty 4: Computer Science

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Identifying fine grained controversy in wikipedia edits of an article.

Master's Thesis

in partial fulfillment of the requirements for
the degree of Master of Science (M.Sc.)
in Web Science

submitted by
Aadil Rasheed

First supervisor: JProf. Dr. Claudia Wagner
Institute for Web Science and Technologies

Second supervisor: Dr. Fabian Flöck
GESIS - Leibniz-Institute for the Social Sciences

Koblenz, March 2019

Statement

I hereby certify that this thesis has been composed by me and is based on my own work, that I did not use any further resources than specified – in particular no references unmentioned in the reference section – and that I did not submit this thesis to another examination before. The paper submission is identical to the submitted electronic version.

	Yes	No
I agree to have this thesis published in the library.	☐	☐
I agree to have this thesis published on the Web.	☐	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐
The source code is available under a GNU General Public License (GPLv3).	☐	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐

.....

(Place, Date) (Signature)

Note

- If you would like us to contact you for the graduation ceremony, please provide your personal E-mail address: .....
- If you would like us to send you an invite to join the WeST Alumni and Members group on LinkedIn, please provide your LinkedIn ID : .....

Abstract

Our work finds the fine grained edits in context of neighbouring tokens in Wikipedia articles. We cluster those edits according to similar neighbouring context. We encode neighbouring context into vector space using word vectors. We evaluate clusters returned by our algorithm on extrinsic and intrinsic metric and compare it with previous work. We analyse the relation between extrinsic and intrinsic measurements of fine grained edit tokens.

Contents

1 Introduction

As a collaboratively edited knowledge database, articles in Wikipedia go through a series of edits. Editing an article involves an editor, creating new revision by addition and removal of string tokens from previous revision of the article. Through these edits, editors reach consensus about the content of the article. Amount of edits after which a consensus is reached on content of an article vary widely across Wikipedia.While for many articles, editors arrive at a consensus after few revisions, for some articles it takes more time to reach consensus. Researchers have grouped lack of consensus across revisions of an article as conflict groups Kittur et al. (2007); Viégas et al. (2004). These models do not take into account fine grained position of edited tokens in their respective revisions. Bykau et al. (2015) groups edits in a Wikipedia article as groups of tokens edited at a fixed location.
We extend Bykau et al. (2015)'s model of fine grained controversy detection using tokens from WikiWho API Flöck et al. and word embeddings from Mikolov et al. TokTrack algorithm used by WikiWho Flöck et al. helps us in localising edits into fine grained neighbourhood. We model fine grained edits into semantic vectors using
embedding vectors from Mikolov et al.. We cluster fine grained edit vectors using the DBSCAN clustering algorithm Ester et al. (1996). We evaluate the clusters from our model against the clusters obtained from Bykau et al. (2015) .
In the followings sections, we state our research question and the relevant related work. We further describe the background knowledge required for our work, which consists of theoretical concepts used in our methodology. We further explain methodology of our algorithm used to address the research question and finally discuss results of the experiments conducted.

1.1 Research Question

The research question addressed in our contribution is, "How can revision history of a given Wikipedia article be grouped as fine grained token level insertions and deletions with its context?".

2 Related Work

Wikipedia, an online free, large scale and multilingual encyclopedia is the result of collaborative edits and contributions to the articles. On one side, where the freedom of editing Wikipedia articles has promoted its accuracy and popularity unlike traditional Encyclopedias which demanded the expertise of domain and supervision of the editors, on the other side, this editing freedom has led to the presence of inaccurate information as well as the absence of accurate information by the intended or ignorant edits by the Wikipedia users Brandes and Lerner (2009).Often such collaborative authoring leads to malicious editing called Vandalism. Chin and

Street studies the revision history of Wikipedia articles at word level and builds content-based statistical models to identify the vandalism by studying the difference between consecutive revisions of Wikipedia articles. They have classified the Vandalism acts in different action categories such as deletion, insertion and change of either text, image, links and formatting tags. Bykau et al. (2015) Vuong and Lim (2008) and Yasseri and Sumi (2012) have considered the text modelling to study Wikipedia controversies and conflicts in the edits. Bykau et al. (2015) has considered the Link modelling of Wikipedia along with the Text model too. Since textual model considers single words as tokens, it facilitates studying the content change in the articles as a sequence of words. We have therefore considered the textual model of Wikipedia articles for our work to capture all the possible information regarding the content alteration in the articles. Revision of an article stands for the version of a document after the changes done by the contributorWikipedia (2005) and Bykau et al. (2015). Brandes and Lerner (2009) models the revision history of Wikipedia articles as visualization graphs without keeping article context in consideration. In such representation, the authors of the concerned article revisions as nodes and the revisions as the edges. Adler et al. (2007) study Wikipedia in terms of the truth worthiness of the articles. Trust values for each Wikipedia articles are computed by a content driven reputation system, where the trust values are determined by the time duration for which the authors contributions live. Longer the contributions live, the higher reputation an author attains and the contribution revert leads to the loss of reputation. The reputation of the original author of the words and the reputation of the authors editing in the proximity of the word are considered for the trust of the word. Vuong and Lim (2008) uses the interaction among Wikipedia contributors to identify controversial articles, rather than analyzing the edit content in the articles. To identify the disputes between a pair of contributors, they consider the words deleted from each Wikipedia articles' edit history. Hence, they have modelled their problem in terms of addition and deletion of words in the way that replacing or editing the content in the article is deleting the old words and replacing them with the new ones.

Bykau et al. (2015) has modelled the controversy in edits to the Wikipedia pages using changes on tokens which are a typical case of "substitution". The reasoning behind this is that any controversy is the result of the alteration of the current content because of the opposing views, rather than the insertion of new content or the deletion of the old content. Instead of identifying substituted tokens per revision, Bykau et al. (2015) identifies the substitution in the neighbourhood of shared tokens. To accomplish this, they fine grain the edited tokens between revisions using the neighbouring context. Once the edited tokens are identified, Bykau et al. (2015) clusters them according to their neighbouring tokens. They hypothesize that these clusters which represent controversy between editors are fine grained at token level. We have based our research work on this idea of identifying tokens in the revision history of an article which are fine grained at more granular level of nearby tokens rather then a set of changed token between revisions. Bykau et al. (2015) further in-

troduces a concept of radius of neighbouring context, which is amount of token used by clustering algorithm to group *changed* tokens into groups. We have extended the notion of context in our research work and have evaluated the context of various length in our work.

3 Background

In this section we describe the theoretical background behind our work. We describe the use of corpus in natural language modeling, we further describe in detail our corpus, Wikipedia. We describe Wikipedia and its collaboration mechanism. We further explain the background behind pre-processing of corpus in general and Wikipedia articles in particular. We explain how WikiWho API from Flöck et al. tokenises each article in Wikipedia and tracking the tokens through the revision history. Next we explain Language modeling and different kinds of language models. Lastly we explain clustering in general and DBSCAN algorithm in particular and different ways to evaluate clustering algorithms.

3.1 Corpus

In natural language modelling, we require a collection of documents on a particular subject in context. This collection of documents is known as corpus. Project Gutenberg Lahiri (2014), Reuters-21578 corpus Lewis et al. (2004), Wikipedia Pasternack and Roth (2008) are some of the famous corpora. Project Gutenberg is the oldest digital library containing a huge collection of around 57,000 free E-books. Reuters-21578 corpus is a collection of articles published on Reuters in 1987. It is indexed according to the articles categories, thus enabling text categorization research.

3.1.1 Wikipedia

Wikipedia, the corpus of our interest, is the world's largest online Encyclopedia, available in multiple languages. Its content can be openly viewed and edited by its readers. Wikipedia has evolved with the contributions of voluntary editors without any Expert supervision. Every visitor can become a Wikipedia Editor. Because of its such nature, it is a good fit for studying the collaborative processes in different aspects like its evolution, content growth, hyperlinks evolution among articles, user reputation, vandalism, controversies detection, collaboration quality and various social aspects of Wikipedia communities Yasseri and Sumi (2012). In Wikipedia different individual editors collaborate in an uncontrolled manner and extend each other's content in order to correct minor mistakes until they reach a consensus Wilkinson and Huberman (2007). Wikipedia records all the previous edits of its pages which is available to its users via history option on the page.
Figure (1) shows a sample Revision history from a Wikipedia page. A user can access a Page's revision history from the time-frame available in the options and is

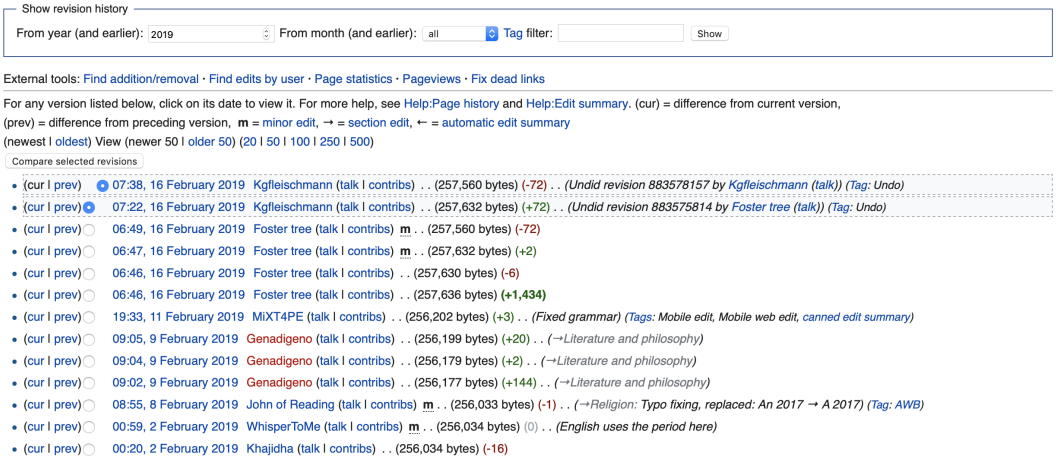


Figure 1: Detail of Revision History of Wikipedia's Germany Page

downloadable as well. Edit summary gives a brief explanation of an edit to the Wikipedia page via links to the saved version, links to the difference from previous version (indicating what has been deleted and inserted in the page). The timestamp of the change, its author and the comments left by the authors about the edits are available. Further, the user can sort the changes according to the timestamp, i.e. both recent and oldest changes. Different options, like finding a particular content addition and removal with the help of the tool called WikiBlame, page statistics and page views are available. Watch list is another option of Wikipedia. Users are notified via Email during any modification to the page they have added to their watch list. This enables the users to fix the undesirable edit done to their page of interest.Wikipedia provides the option to view the difference in the content of the page across various revisions. Wikipedia articles are of high quality inspite of the challenges of open edit and collaboration facility Giles (2005). Forte and Bruckman (2005) hold that such quality is the result of peer recognition similar to an academic community.

3.2 Tokenization

Representation of corpus content is required to be able to become the subject of language modelling. Documents in a corpus are composed of the articles, paragraphs, sentences, phrases, words and characters, which are the atomic representations of those documents. We need to segment the text in the corpus of documents into atomic representation before its modelling and thus convert the documents content into a sequence of individual occurrences of smallest meaningful linguistic units called tokens. Depending upon the language modeling task at hand we choose different representation and every content entry is converted to a sequence of the tokens as per the chosen representation. Unique tokens present in a corpu' docu-

ments as per the representation chosen build up the vocabulary of the corpus. For example, if words are chosen as tokens, in that case, total unique words in the corpus become the vocabulary of the corpus.
One of the most popular representation of English Wikipedia articles is "words", as words are smallest semantic representation of English language documents Bykau et al. (2015)Vuong and Lim (2008)Yasseri and Sumi (2012)Flöck et al.. The smallest unit of undivided string is called token. The tokens can be words or links to other pages or entries in the Wikipedia articlesBykau et al. (2015).

3.3 WikiWho API

We need to choose a token representation scheme of Wikipedia for our work and Flöck et al. WikiWho API provides such pre-processed representation. We have chosen WikiWho API because: (1)WikiWho tokenises the text content of a revision where tokens $\approx$ words. We use WikiWho API to obtain list of string tokens created by tokenisation of article's content in revision R_i in Equation (1). Words, punctuation and special Wikipedia symbols (e.g. '[[') are seen as separate unique tokens and tracked independently. (2) WikiWho API provides the tokenised earliest known edit history and change information of Wikipedia articles in various languages through revisions and has released the dataset called Toktrack dataset in the same context. WikiWho returns the revision history of a given article consisting of ordered set of revision instances and tracks the tokens accurately across revision history with an accuracy of around 95%. Each revision $R_i \in Revision\_history$ has text content of article in that revision as well as the meta information about its creation. (3) It tracks the token changes across revisions, which we aim for in our work using WikiWho algorithm and maintains in and out lists based on the token deletion and insertion in an article's revision history. Corresponding to each revision $R_i \in Revision$, WikiWho returns a list of string tokens $string\_tokens_i$ ordered by their position in its text content. WikiWho API returns a list of unique integers token $wiki\_who\_tokens_i$ corresponding to each $string_token \in string\_tokens_i$ too. These tokens uniquely track each string token in the string content of the article. Formally,

$$r_i = (id_i, editor_i, t_i, string\_tokens_i, wiki\_who\_tokens_i) \tag{1}$$

A revision $R_i \in Revision\_history$ contains a unique revision id id_i, an editor e_i, a timestamp t_i a list of tokenised string tokens $string\_tokens_i$ and a list of WikiWho tokens $wiki\_who\_tokens\_i$. WikiWho API gives both the string tokens and WikiWho tokens which are ordered according to their positions in Wikipedia article.
WikiWho token information is divided into three categories, namely the current_content, deleted_content and revisions per article. (1) current_content contains all the tokens of all the articles present in the last revision. current_content is represented by the fields such as $page\_id, last\_rev\_id, token\_id, str$, $origin\_rev\_id, out, in.$ current_content maintains the provenance of the tokens in the revisions using the "in"

and "out" lists. (2) deleted_content contains the tokens which were present in at least one revision and not present in the last revision. (3) revisions contains all the revisions of the articles in order of their processing by WikiWho. Revisions are represented by $page\_id, rev\_id, timestamp, editor$. Formally,

$$Revision\_history = \begin{bmatrix} r_1 & r_2 & r_3 & \cdots & r_i & r_{i+1} & \cdots & r_n \end{bmatrix} \tag{2}$$

There are two lists of revisions, namely, in and out lists. in list is the list of revisions, where the tokens are reinserted after at least one deletion previously. out lists contains containing revisions, where the tokens are deleted. Inserting the token again after at least its one deletion is called $reinsertion$.

3.4 Language Modelling

Natural language modelling aims to determine the regularities of natural language and estimate the distribution of language linguistic units like words, sentences, documents Lau et al. (1993); Rosenfeld (2000). Application of Language Models can be seen in various tasks, like speech recognition, machine translation, spelling correction etc Collins. To fulfill the goal of language modelling, an efficient representation of tokens is necessary. The type of representation chosen at hand depends on the target task. In the following section, we describe each such token representation and how some of these representations are relevant to our work.

3.4.1 Vector space modeling

Salton et al. (1975) Manning et al. (2008) Vector space modelling is a technique in natural language processing to represent tokens or simply, the terms/words of the documents in a fixed dimensional vector space. Each document in a corpus, like an article in Wikipedia, containing total n terms in the vocabulary, is represented by an n dimensional vector in the vector space r_n. Weights are assigned to each term in the document vector which is a measure of its importance and the weights are therefore zero for the terms not present in the document. There exist different kinds of representation methodology and term weighting schemes in Vector Space model.

Bag of words model Bag of words model is one of the text representation methods in language modelling. It is called "bag of words" because the representation is done irrespective of the order of occurrence as well as the semantic importance of the words Manning et al. (2008); Zhang et al. (2010); Sivic and Zisserman (2009). Consider the following two sentences of a Wikipedia article:

1. "It was expected that he will come. He did not come."
2. "He could have come."

The text in the article is tokenised.

1. $[$"It","was","$expected$","$that$","he","$will$","$come$","He","did","not","$come$"$]$
2. $[$"He","$could$","$have$","$come$"$]$

Multiple occurrences of the same token are dropped and unique tokens build up the vocabulary of the model and the article is represented by the vector:
$[$"It","was","$expected$","$that$","he","$will$","$come$","did","not","$could$","$have$"$]$
Each sentence vector is represented as the vector of the same size as that of the vocabulary and each entry in the sentence vector correspond to the count of the corresponding entry in the vocabulary of the article. Word count track the occurrence frequency of the tokens in the vocabulary. The representation results as follows:

1. "It was expected that he will come. He did not come." is represented by the vector [1 1 1 1 3 1 3 1 1 0 0]
2. "He could have come." is represented by the vector [0 0 0 0 1 0 1 0 0 1 1]

Though Bag of Words model is easy to understand and implement, vectors size increases as the vocabulary size increases. The semantic meaning of the tokens is not known as the tokens of the same meaning can be modelled as differently even if they reflect the same concept. The spatial information between the words which can be related to the words' order or proximity is lost Sivic and Zisserman (2009).

One-hot encoding Lantz (2013); Manning et al. (2008) One-hot encoding is a representation as well as a term weighting scheme , similar to the Bag of Words model which follows a Boolean vector approach. One-hot encoding represents each term on n independent dimensions, where n is size of vocabulary. If the term is present in the document, it is assigned a weight value of 1 otherwise 0 unlike the term frequency in Bag of Words model.

Tf-Idf Tf-Idf is a common mechanism of term weighting often used with different vector space models scheme. It weights the each vector by using frequency statistics of occurrence of a term in a document and in the entire corpus Manning et al. (2008); Ko (2012); Jurafsky and Martin (2000). Tf stands for term frequency and Idf stands for inverse document frequency. Tf, as the name suggests, counts the frequency of a term in a document/article, while Idf measures the importance of the term in the whole corpus and measures how rare the word is in the entire corpus. Tf-Idf is obtained by the product of Tf and Idf. Idf is obtained by log(m/df_i), where m is the no. of total documents in the corpus and df_i indicates the document frequency in which the term i is present.

Formally, Inverse Document Frequency, idf is:

$$
\begin{aligned}
&idf(t, C) = log\frac{N}{d_t}, where \\
&t \text{ stands for the term,} \\
&C \text{ stands for the Corpus,} \\
&N \text{ is the total number of documents in the corpus } C, \\
&d_t \text{ are the number of documents } d \text{ in which the term } t \text{ appears}
\end{aligned}
\tag{3}
$$

The higher is the weight assigned to a term by Tf-Idf, rarer the term is as the term might occur more in small number of documents in the corpus. Likewise, smaller Tf-Idf indicates the term is more common because the term might be either occurring fewer times in a single document or either occurring in many or all documents of the corpus. One of the advantages of this weighting scheme is that it relies on term frequency for the weighting scheme. But, like Bag of Words and One hot encoding, Tf-Idf lacks in measuring the semantic similarity among the terms as it strives on the word count irrespective of their semantic importance and thus, documents of similar subject but different terminology cannot be correlated.

3.5 Language Models

3.5.1 Statistical Language Models

Statistical language modelling employs statistical estimation methods to determine the distribution of a sequence of words in a language Rosenfeld (2000). Each textual unit, be it words (tokens), sentences or documents are modelled as a random variable with some probability distribution Rosenfeld (1996). An intuitive way to model a sequence of tokens is using the joint probability rule which results into the product of conditional probabilities Rosenfeld (1996, 2000). The chain rule of conditional probability results to:

$$
\begin{aligned}
P(S) &= P(t_1, t_2, ..., t_n) \\
&= P(t_1)P(t_2 \mid t_1)P(t_3 \mid t_2, t_1)...P(t_n \mid t_1, ..., t_n - 1) \\
&= \prod_{i=1}^{n} P(t_i \mid t_i - 1)
\end{aligned}
\tag{4}
$$

Statistical language models tend to estimate $P(t_i \mid t_i - 1)$ where $t_i - 1$ is often known as history. The most naive way to estimate the probability of a sequence of words or tokens, say for example, $P(t_5 \mid t_4, t_3, t_2, t_1)$, would be to use counting principle.

$$
P(t_5 \mid t_4, t_3, t_2, t_1) = \frac{count(t_1, t_2, t_3, t_4, t_5)}{count(t_1, t_2, t_3, t_4} \tag{5}
$$

Counting instances as in Equation (5) is practically not feasible because of the huge amount of data required to learn it and the goal of learning the joint probability function of sequences of words leads to the problem of curse of dimensionality. This is because the trained language will be tested on the test data which might contain different word sequences than training data. Thus huge training examples are required to learn and differentiate various combinations(or sequences) of wordsBengio et al. (2003).To address this problem, Markov model assumption can be used which states that the future word can be predicted based on its relative short history, composed of couple of previous words. Formally,

$$P(t_1, .., t_n) \approx \prod_i P(t_i \mid t_i - k, ..t_i - 1), \text{where}$$
$$\text{k is the number of previous tokens which are considered} \tag{6}$$

In one of the earlier works, Kuhn (1988) used the same Markov assumption in speech recognition systems. Below we describe the models called N-Gram models based on the Markov assumption which address the problem of curse of dimensionality as described above. They present an approach to tackle the curse of dimensionality problem by generalizing to the short sequences of words in the training data according to the word order.

N-Gram model N-Gram models are the language models which assign probability to contiguous n sequence of words in a given text. N-gram model estimate the future state based on the present state. The future state refers to the probability estimate of the next word, n and present state means the context which is the present sequence of the words, i.e $n - 1$ words. N-gram models strive on the semantic importance of the immediate preceding neighbours of the current word as closer words in a word sequence tend to be statistically more dependent. Bahl et al. (1983); Brown et al. (1992); Bengio et al. (2003). An N-gram model is a Unigram model when n=1, Bigram when n=2, Trigram when n=3 and so on.

$$P(t_1, .., t_n) \approx \prod_i P(t_i) \tag{7}$$

$$P(t_1, .., t_n) \approx \prod_i P(t_i \mid t_i - 1) \tag{8}$$

Equation (7) refers to the Unigram model formulation where n = 1 and it considers only one word. Equation (8) is the Bigram model formulation which conditions each word on the previous word. Similarly, in Trigram model conditions on the previous two words, 4-gram model conditions

3.5.2 Neural Language Models

The major problem with Statistical language models is that they lack in capturing the semantic similarity between words. For example, Having seen the sentence, "He is eating Pizza" during training should enable the model generalize the sentence, "He is making Pasta" almost as likely because "Pizza" and "Pasta", have similar semantic role (of belonging to the category Food) and similarity, the actions, "eating" and "making". Another problem in obtaining the probability of the next word using N-gram model does not take into account context more than n words Bengio et al. (2003).
Neural language models were first introduced in Bengio et al. (2000, 2003). They propose a language model based on neural network in which represent each word in the vocabulary by a feature vector. Such feature vectors are distributed representation of words in the vector space. Distributed representation of word is a feature vector which identifies the meaning of the word and is not mutually exclusive. Each word is mapped to point in the vector space. The idea is that semantically similar words lie closer to each other in this vector space, thus enabling the transformation of a sequence of words to a sequence of learned feature vectors. Such representation reduces the impact of curse of dimensionality by allowing the model to generalize to the unseen combinations of words in terms of their similarity in the feature representation.A feed forward neural network architecture is used with a linear projection layer and a non-linear hidden layer to learn jointly word vector representation as well as statistical language model. The importance of neural language models w.r.t our work lies in our idea to capture the semantic similarity of the fine grained edits across different revisions. We describe more behind this approach in the following section.

Word2vec Mikolov et al.Word2vec is a popular approach of representation of word sequences. Word2vec uses the concept of nearby words, i.e. word vectors which are semantically similar lie close to each other in the vector space. Skip gram is one of the model architecture which uses Word2vec for word representation. It predicts the words semantically similar based on the current input word within some range before and after the current word. The number of words before and after the current word to be considered as the context words builds up the context window. Typical size of the context window is 5, i.e 5 words before the current word and 5 after the current word. The network is trained on the input combinations of the words in the training vocabulary according to the window size. For classification purpose, the input to the neural network are one-hot encoded vectors of the input word of the same size as that of the vocabulary used in training the network. Its vector representation works by picking the words in certain range nearby to the each current word in the sentence randomly. It predicts the similarity of those picked words by outputting the probability of the context similarity using Softmax function. Bojanowski et al. (2016) and Joulin et al. (2016) improves the word2vec from Mikolov et al. to

create fast-text model, which accounts for sub-words of a word and is trained on Wikipedia.
Skip gram model of train word vectors where embedding is hidden projection of probability of predicting the word given the neighbourhood context is able to capture the semantic similarity between string tokens Mikolov et al.. Skip gram word vectors perform well on series of tasks such as analogy in words as well as text retrieval, classification and clustering. Our motivation in using Skip gram model lies in using pre-trained Word Embeddings from Mikolov et al. with distance metric to calculate the similarity in the set of tokens in neighbouring context with accountance of linguistic similarity of tokens in neighbourhood. Whereas Bykau et al. (2015) uses Jaccard similarity in the set of tokens in neighbouring context and Jaccard similarity on tokens compares occurrence of exact word without accounting for linguistic similarity of words in neighbourhood.

3.6 Clustering

Clustering is an unsupervised machine learning technique which identifies pattern in given data points without being trained on explicit labels for each group. Being an unsupervised learning algorithm, it only has the unlabeled inputs without any corresponding target variables and the task is to discover interesting patterns in the data without the explicitly provided labelled outputs. Therefore, there is no specic error metric to evaluate the model performance since we are not provided with labelled target outputs unlike supervised learning algorithms such as classification, in which we can compare the model's predicted outputs with the ground truth labels. In clustering, given a set of inputs, $\{x_1, x_2, \cdots, x_n\}$ where every input x_n is characterised by a set of attributes, i.e. $x_i = \{a_1, a_2, \cdots, a_m\}$, the task of function γ is to group similar inputs together and these groups are known as clusters. Thus, we need to nd the clusters, $\Omega = \{w_1, w_2 \cdots w_k\}$, and we find the mapping function:

$$\gamma : x_i \rightarrow \Omega_i \tag{9}$$

The goal of clustering is to find the clusters such that similar groups contain the data points as similar as possible and dissimilar groups contain the input points as dissimilar as possible.
For data to be clustered in groups a similarity measure needs to be defined between them. Generally similarity is defined as a distance metric. Any function $d(a, b)$ between two points a and b is called metric if it follows the following three properties.

$$d(a, b) = d(b, a) \tag{10}$$

$$d(a, b) > 0, \text{ and } d(a, a) = 0 \tag{11}$$

$$d(a,c) <= d(a,b) + d(b,c) \tag{12}$$

Few example of metric are Minkowski distance, cosine distance, hamming distance etc. There are few distances which only follows 10 and 11 but not the 12 also called triangle rule. One such distance metric is Jaccard distance.

3.6.1 Types of Clustering

Clustering tasks can be divided into two major categories based on the criteria chosen. The two types of criteria chosen for the clustering task can be either cluster types and the function γ.

1. Based on the types of clusters: By the types of clusters, we mean the cluster formation based on the relation of the clusters to each other.Clustering can be categorized as follows based on the types of cluster formation:
   a) Flat Clustering: Flat clustering relates to the cluster formation in a way such that there is no explicit relation of clusters to each other.

      K-means clustering is a famous flat clustering algorithm which partitions the input data into k partitions where each partition belongs to one cluster. The similarity is based on the closeness of the data points in a cluster to the centroid of the respective cluster. The closeness of a data point to the centroid of the cluster is measured by the Euclidean distance metric. K-means algorithm is a fast and efficient algorithm because of its simple implementation and less complexity, O(n) for each iteration as the number of iterations are usually small. A big disadvantage of K-means is that it is necessary to know the number of clusters, i.e. K beforehand which is often difficult to determine. DBSCAN (Density based spatial clustering of applications with noise) is yet another important flat clustering algorithm. In this algorithm, the input data points are partitioned in different density regions. Data points are grouped together in high-density regions to form clusters and the outliers are marked by the points lying in low-density regions.

   b) Hierarchical Clustering: Hierarchical clustering leads to the cluster formation in a hierarchy. Before clustering algorithm identifies explicit hierarchy of relations among points to be clustered. It uses two kind of order to construct hierarchy among data points, a bottom-up approach or top-down approach in cluster formation.

2. Based on the mapping function Γ: The cluster formation depends on the definition of the mapping function Γ too. Based on this, there can be:
   a) Hard clustering assignment: Each data point belongs exactly to only one cluster among all the clusters formed, thus enabling a hard assignment.

b) Soft clustering assignment: The data points may belong to more than one cluster, leading to their assignment to multiple clusters with some probability.

3.6.2 DBSCAN Clustering

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Ester et al. (1996). It uses the concept of core points to identify regions of high density and low density using neighbourhood radius of eps. DBSCAN has two parameters, Minimum sample size $min\_samples$ and distance of eps. eps distance is the maximum distance two points can be apart to be considered in same group and $min\_samples$ is minimum number of points in the group which has to be at least eps apart for the point to be considered core point of the group. Algorithm works by identifying few core points than keeps adding more points to its cluster and leaves remaining point as outliers.
Each point p_i belonging to set of point to be clustered is put into group of core point if it has at least $min\_samples$ number of point which are less than or equal to eps distance from p_i. After identifying core points, If any other point q_i lies at the eps distance to any p_i belonging to core point its added to the cluster. Thus all the reachable points from core points are put into the same cluster as core point, but for a point to be a core point it has to be in high density region where at least $min\_samples$ points are with in eps distance to it. As the algorithm is density based it created spatial clusters of variable sizes and is sensitive to choice of parameters, eps and $min\_samples$. Amount point not clustered in any group is also dependent upon choice of eps and $min\_samples$.

3.6.3 Evaluation

Groups in clustering are created in an unsupervised manner, so each group does not have predefined meanings. In order to evaluate the quality of cluster groups identified by the algorithms, one way is to use the metric which evaluates the group without taking help of user created groups. Such metric which evaluates clustering just using intrinsic quality measure are called intrinsic evaluation metric. The other way to evaluate a clustered group is to take help of external annotators. Metric utilising external annotators is called extrinsic metric.

Intrinsic evaluation Intrinsic evaluation criteria relies on the quality of clusters. Quality of clusters is determined majorly by inter-cluster and intra-cluster similarity among the identified clusters. The goal is to have higher intra-cluster similarity, indicating the enclosed data points in the cluster are similar and lower inter-cluster similarity, indicating the data points in different clusters are dissimilar. Intrinsic cluster measures try to quantify the quality of grouping based on similarity measure which leads to more coherent clusters. As most of the clustering algorithms

have some similarity measure to create the groups in first place, it is important to use the intrinsic evaluation metric carefully in order to account for implicit bias for similarity measures in clustering algorithms. Intrinsic clustering mechanism provides us with a notion of similar and dissimilar cluster which can be used to choose an algorithm over other an help us gain insight into the process of clustering.
One such intrinsic evaluation scheme is the Fowlkes-Mallows index(FMI) E.B. Fowlkes (1983). It measures the similarity between two clustering generated by two algorithms and is defined as the geometric mean between of the precision and recall: Formally,

$$FMI = \frac{TP}{\sqrt{(TP + FP) * (TP + FN)}}, \text{where} \\ \begin{aligned} TP &\text{ is number of true positives,} \\ FP &\text{ is number of false positives,} \\ FN &\text{ is the number of false negatives} \end{aligned} \tag{13}$$

Fowlkes-Mallows index score of 1 denotes perfect agreement while a score of 0 denotes no agreement.

Extrinsic evaluation Extrinsic evaluation on the other hand gives us a measure of cluster which is not solely based on notion of similarity of data points to be clustered but is dependent upon notion of similarity created by humans. It uses external knowledge of labels to help humans annotate with the labels. This leads to creation of ground truth of groups of clusters on which the algorithm created clusters are evaluated. Extrinsic evaluation metrics' rate the algorithm created clusters according to its agreement with human created ground truth groups.
VMeasure is an entropy based extrinsic evaluation measure which evaluates a cluster created by an algorithm against the annotated class label from human annotators. It uses measures of Homogeneity and Completeness to calculate entropy in clustering labels assigned by algorithm. Homogeneity quantifies how many of the data points in each cluster belong to same class labels prepared by human annotators. Similarly, completeness quantifies how many class labels created by human annotators belong to the same cluster. Therefore, Homogeneity is equivalence to precision of clustering algorithm to put same class labels in one cluster group, while completeness is ability of clustering algorithm to be able to recall all class label in one cluster. We can see both are symmetrical and competing metrics, so Vmeasure is defined as harmonic mean of Completeness and Homogeneity.

4 Methodology

In this section we formulate the methodology to answer our research question of identifying fine grained inserted and deleted tokens in the revision history of an article. First, we explain the problem of identifying fine grained edits in context where context lies in the neighbouring tokens. We further give related definitions and formulas. We also present the overall architecture of our methodology. Later, in the subsections, we describe in detail the steps leading to identification of fine grained edits, which we call change objects. In the last subsection, we devise methodology to create groups of change object, evaluate it and compare our model with model from Bykau et al. (2015).
In order to identify groups of inserted and deleted text in the $Revision\_history$ of a given article which were similar in its meaning in the document, we group inserted and deleted tokens according to their neighbouring context in respective revisions. For example, let us consider an article whose $Revision\_history$ consists of four revisions.

1. The sister sings songs
2. The sister sings nice songs
3. The sister sings songs
4. the small bird sings songs

Upon observing changes in content of article between revisions we see that the word “nice” is inserted in revision 2 and is removed in revision 3. In revision 4, the text “small bird” replaces the word “sister”. Since the insertion of word "nice" and its removal share context in terms of neighbouring tokens ("sings" in the left, and "songs" in the right), we say that they belong to same group of changes. Similarly, the neighbouring context for the removed word “sister” is the same as the context of the inserted words “small” and “bird”, thus they also form their own group. Although the words “sister”, “small” and “bird” are semantically different, they belong to the same group because they share the same context. Therefore, the unchanged neighbouring context is responsible for deciding which objects belongs to the same fine grained *changed* group, i.e. the change object.
Our objective consists in clustering these change objects across multiple revisions, and not only of contiguous revisions as the toy example above. We can use the same logic of the neighbouring context to detect similar change objects. However, since the text in the article evolves with revisions, the neighbouring context will not be exactly the same, and the clustering should be done using a similarity measure. Bykau et al. (2015) uses Jaccard similarity in the set of tokens in neighbouring context. Jaccard similarity on tokens compares occurrence of exact word without accounting for linguistic similarity of words in neighbourhood, hence we use pre-trained Word

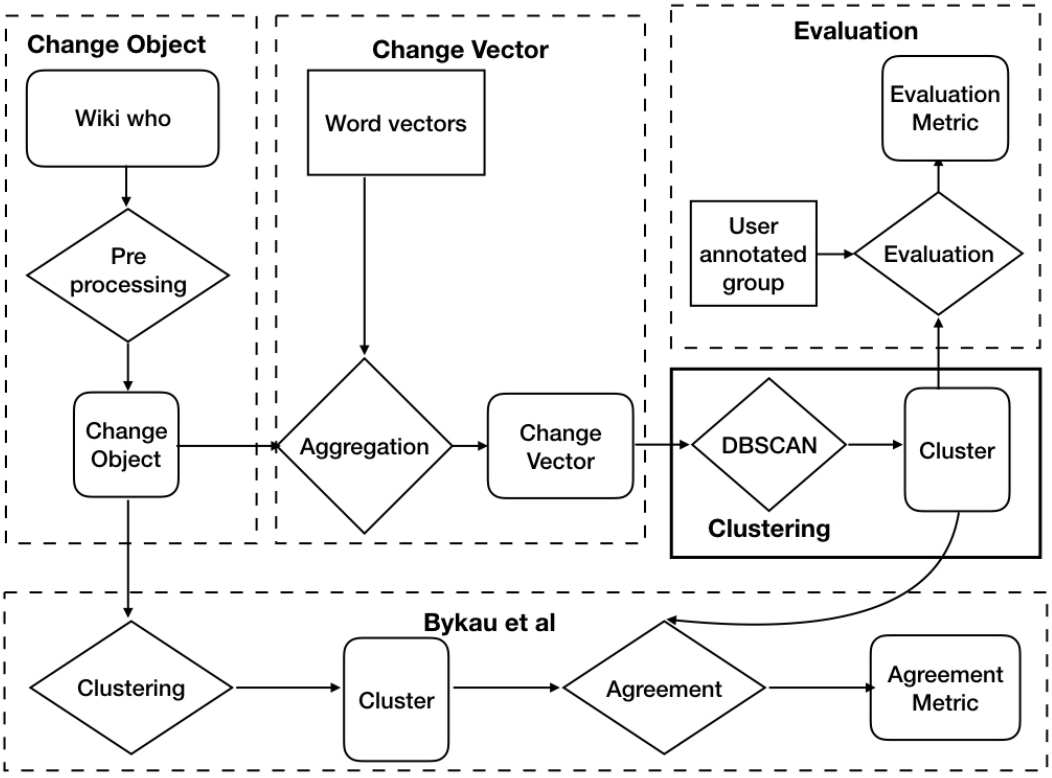


Figure 2: System Architecture

Embeddings from Mikolov et al. with distance metric to calculate similarity, as described in Section (3).

The entire system architecture is summarised in Figure 2. It presents the overview of the flow of data starting from tokens obtained from the WikiWho API and ending in the evaluation of the clustering of the change objects.

The first step in the architecture in 2, Change Object is the identification of fine grained change objects. A change object comprises the gap of inserted and deleted tokens between two consecutive revisions along with their shared left and right context. While Bykau et al. (2015) defines the gap as a substitution, i.e. there must be at least one insertion and one deletion from same neighbouring context, we allow any one of the inserted or deleted token set to be empty.

The second step in the System architecture in 2, Change Vector is the encoding of context of change object as a fixed dimensional vector, called change vectors. Before clustering change objects which share a similar context neighbourhood, we project the context neighbourhood of all change objects identified in the previous step in a fixed dimensional vector space. This fixed dimensional vector space is necessary for the clustering algorithms as a distance metric needs to be defined on all the objects

to be clustered. Change vectors of fixed dimension allows us to efficiently calculate distances between change objects.
The third step in the system architecture in 2, is the Clustering. We identify groups of change objects which have high similarity in their neighbourhood context. We do this by using the **ChangeVector** that characterise each Change Object.
In the fourth step, we create clusters for different sets of parameters and do intrinsic and extrinsic evaluation on clusters. Intrinsic evaluation is done to identify stability in clusters identified by our algorithm while extrinsic is done against a ground truth prepared by human annotator.
In the fifth and the final step, we implement Bykau et al. (2015) and compare it with our algorithm.
In the following subsection, we describe each step of the architecture in Figure 2 in detail, starting from the identification of Change Object, followed by the creation of **ChangeVector**s , and at last clusering and the intrinsic and extrinsic evaluation of Change Objects.

4.1 Change Objects

As shown in the architecture diagram 2, in order to find groups of fine grained change object we first need to group change tokens of an article across its revision history into change objects. To identify change objects of an article, we process the revision history $Revision\_history$ of an article's content. Each revision $r_i \in Revision\_history$ of a given article is retrieved from the WikiWho API. As discussed in the background section, WikiWho API returns the tokenised strings of each revision content of an article. We identify gaps of tokens which change between two consecutive revisions r_i and r_{i+1} which are bounded by the same tokens in the neighbouring context. Figure 3 below shows the three revisions of the article. Wikiwho tokens can be tracked uniquely (with 95% accuracy) across revisions of a given article. For example, notice that in the Figure 3 , in revision r_2, the string "." comes at two places but WikiWho tracks them separately and assigns them unique token of 18 and 20. In this case, Wikiwho would be able to understand that token 18 belongs to the end of the revision due to the shared context, i.e left token 16 ("For") and the left end of document token (not shown).

Two consecutive revisions can be described by multiple edits, as not all the changes between revisions share the same text, i.e. they can happen in different parts of the article. Therefore, we define a set of $Edits$ between each consecutive revisions $r_i, \quad r_{i+1}$, representing the edit process. Hence, there exists an ordered set of edits $Edits$ for an ordered set of $Revision\_history$. Formally, set of $Edits$ can be defined as follows:

$$Edits = e_{<1,2>}, e_{<2,3>}, \cdots, e_{<i,i+1>}, \cdots, e_{<n-1,n>} \tag{14}$$

id - 0

String Tokens	A	tree	was	standing	Close	!
WikiWho Tokens	10	11	12	13	14	15
Pos_rev	0	1	2	3	4	5

id - 1

String Tokens	A	tree	was	standing	Close	!	For	real	.
WikiWho Tokens	10	11	12	13	14	15	16	17	18
Pos_rev	0	1	2	3	4	5	6	7	8

id - 2

String Tokens	A	tree	was	standing	nearby	.	For	sure	.
WikiWho Tokens	10	11	12	13	19	20	16	21	18
Pos_rev	0	1	2	3	4	5	6	7	8

Figure 3: Set of revisions

Each edit $e_{<i,i+1>}$ denotes the process of editing, i.e. an editor $editor_{i+1}$ sees an existing revision r_i, and proceeds to remove a set of tokens $deletion_i$ (from r_i) and insert another set of token $insertion_{i+1}$ (into revision r_{i+1}). Formally,

$$e_{<i,i+1>} = (deletion_i, insertion_{i+1}), \quad \forall e_{<i,i+1>} \in Edits \tag{15}$$

With the the set of three revisions returned by WikiWho API shown in Figure 3 as an input, we can identify the edits $e< 0, 1 >$ and $e< 1, 2 >$, i.e. the changes between consecutive revisions. From WikiWho API, for each revision r_i we can tell which of tokens ($wiki\_who\_tokens_i \in r_i$) will be removed in the next edit $e_{<i,i+1>}$, and which tokens were added in the previous edit ($e_{<i-1,i>}$). Leveraging this information given by WikiWho API, we retrieve sets of $deletion_i$ and $insertion_{i+1}$ corresponding to each edit $e_{<i,i+1>}$.
In our example of 3 revisions in Figure 3, $e_{<1,2>}$ consists of removal of tokens “close”, "!" and "real" and addition of tokens “nearby”, "." and “sure”, i.e. $deletion_1 = \{\text{1}close\text{J}, "!", "real"\}$ and $insertion_2 = \{\text{1}nearby\text{J}, ".", \text{1}sure\text{J}\}$. Sets of insertions and deletions are replaced by their WikiWho token counterparts and re-written as $deletion_1 = \{14, 15, 17\}$ and $insertion_4 = \{19, 20, 21\}$, as WikiWho tokens uniquely identify words (e.g. it distinguishes between the two "." characters). We always use $wiki\_who\_token$ in our Algorithm 1 to identify change object because its unique tokens helps us in avoiding all the ambiguity in identifying tokens uniquely.
Edit $e_{<i,i+1>}$ contains all the inserted and deleted WikiWho tokens in two separate sets. The deletions set $deletion_i$ is obtained from r_i, whereas the insertion set $insertion_{i+1}$ is obtained from r_{i+1}. Each edit is divided in several fine grained change objects; two inserted or deleted tokens belong to the same change object if they are in they are contiguous. The table 6 shows the fine grained change objects that can be obtained from table 3.

Formally, a change object is described by:

$$Change\_object = (e_{<i,i+1>}, left\_context, gap, right\_context) \tag{16}$$

Here, gap is the inserted and deleted tokens set, the $left\_context$ represents the tokens that appear before the gap and $right\_context$ represents the tokens that appear after the gap. In theory, $left\_context$ extends all the way to the beginning of the document, and the $right\_context$ all the way to the end; in practice, a cutoff is necessary for computational reasons. For example, a change object of context 5 means that there are only 5 tokens on the left and on the right of the change object. The immediate left and right context, called $left\_context\_start$ and $right\_context\_start$ respectively, are important in order to identify the contiguous insertions and deletions.

Input					Output	
Edit	Pos_rev_i	token	Pos_rev_ i+1	To rev	left_token _start	right_token _start
<0,1>	-	For	6	1	!	
<0,1>	-	real	7	1	!	
<0,1>	-	.	8	1	!	
<1,2>	-	Nearby	7	1	standing	For
<1,2>	-	.	8	2	standing	For
<1,2>	-	sure	9	2	For	.

Figure 4: Context of insertions

4.1.1 Contiguous insertions and deletions

A token always has two neighbors: one on its left side and other on its right side. A special token marker is used to indicate if the token is present at the start or at the end of the article.

Formally,

$$\forall token_k \in (insertion_{i+1}, deletion_i) \implies \exists(left\_context\_start, right\_context\_start) \tag{17}$$

such that $left\_context\_start$ is the first unchanged token to the left of $token_k$ and $right\_context\_start$ is the first unchanged token to the right of the $token_k$.

Assuming that we have identified the $left\_context\_start$ and $right\_context\_start$, it is provable that tokens which share $left\_context\_start$ and $right\_context\_start$ in an edit $e_{<i,i+1>}$ must come at contiguous positions bounded by $left\_context\_start$ and $right\_context\_start$. This can be proved by contradiction below:

PROOF Let there be two tokens $t_j, t_k \in wiki\_who\_tokens_i$ which share $left\_context\_start$ and $right\_context\_start$ in edit $e_{<i,i+1>}$.
$pos_j < pos_k$ where pos_j is the position of t_j and pos_k is the position of t_k, but they are not part of contiguous edits.
Since t_j and t_k are not part of contiguous edits, it follows that there must exist at least a single token t_p such that $pos_j < pos_p < pos_k$ which remain unchanged between revisions $R_i, R_{i+1} \in e_{<i,i+1>}$.

Input					Output	
Edit	Pos_rev_i	token	Pos_rev_ i+1	To rev	left_token _start	right_token _start
<1,2>	-	close	5	2	standing	for
<1,2>	-	!	6	2	standing	for
<1,2>		real	8	2	for	$

Figure 5: context of deletions

Since t_j, t_k share $left\_context\_start$ token then $left\_context\_start$ token should be the first token before t_j, t_k which is unchanged in $e_{<i,i+1>}$. Same can be likewise proved for the $right\_context\_start$.
Hence t_p does not exist proving that all the pair of token which share $left\_context\_start$ and $right\_context\_start$ must be part of a contiguous edit in $e_{<i,i+1>}$.
Those, once we have identified $left\_context\_start$ and $right\_context\_start$ for each token in $e_{<i,i+1>}$ (next section), we can divide $insertion_{i+1} \in e_{<i,i+1>}$ into change objects, each of which contains the inserted and deleted tokens with same left and right context, called gap of the change object. Each token in $insertion_{i+1}, \in e_{<i,i+1>}$ which has same $left\_context\_start$ and $right\_context\_start$ belongs to the same group of fine grained inserted gap bounded by right and left context. For example, the Tables 4 and 5 shows the $left\_context\_start$ and $right\_context\_start$ for all insertions and deletions of the example in Table 3.

The Table 4 presents all the inserted tokens that correspond to the revisions given in Figure 3. The left side of the Table (Input) shows the corresponding edit $e_{<i,i+1>}$, the position of the token in r_i (empty as they did not exist in that revision), the text of the token, and the position of the token in $r_i + 1$. The right side of the table (Output) shows the left and right neighbouring token. The insertion of the tokens $\{"For", "real", "."\} \in e_{<0,1>}$ have the same $left\_context\_start$ (“!”) and $right\_context\_start$ (empty, or end of document marker). The tokens $\{"nearby", ".", "sure"\} \in e_{<1,2>}$ are divided into two change objects based on their $left\_context\_start$ and $right\_context\_start$ tokens. The first group consists of "$nearby$","." with "standing" as their $left\_context\_start$ and "For" as their $right\_context\_start$. The second group has just one token "$sure$" with "For" as its $left\_context\_start$ and "." as its $right\_context\_start$. Thus, in total we have obtained three fine grained insertion units out of two edits ($e_{0,1}$ and $e_{1,2}$) using the context of their respective revisions.
Similarly, the Table 4 presents all the deleted tokens that correspond to the revisions given in Figure 3. Once we detect the $left\_context\_start$ and $right\_context\_start$,

Edit	Left_context	insertions	deletions	right_context
<0,1>	!	For, real, .		
<1,2>	standing	nearby, .	Close, !	for
<1,2>	For	sure	sure	.

Figure 125. List of change objects.

Figure 6: Edit $e_{<1,2>}$

we can use the same procedure to group the tokens into the fine grained deletion units, in this case two units are found: "$close$","!" and "for". Notice,however, that the context is extracted from revision r_i (instead of r_{i+1}); this is because the token does not exist in r_{i+1} since it was deleted between r_i and r_{i+1}. This is evident by looking to the position columns (pos_r_1 and pos_r_1). For insertions, pos_r_1 is empty, whereas for deletions, $pos\_r_1 + 1$ is the one empty.
Finally, the fine grained insertion and deletion units can be associated when the units share the same $e_{<i,i+1>}$, and $left\_context\_start$ and $right\_context\_start$ in order to obtain the change objects. For example, the fine grain insertion unit "$nearby$","." can be associated with the deletion fine grained unit "$close$","!". Figure 6 shows the complete list of change objects generated from the example in Figure 3. Here, we distantiate from Bykau et al. (2015), as already described in the brief introduction of Section (4), we do not discard insertions or deletions which do not have an associated counterpart. For example, the first change object in Table 6 does not contain any deletions, just insertions.

4.1.2 Identifying left and right context

In the previous section, we assumed that we had identified the $left\_context\_start$ and $right\_context\_start$; in this section we present the procedure to actually identify them. The procedure, inspired from the Fourier Transform Bracewell (1978), is not only efficient but it also provides a direct way to identify the change objects with this procedure. We specifically design a transformer like discrete auto correlation convolutions transformer, which tries to find auto correlation in insertions and deletions to identify contiguous edits.
To mark the starting position of left and right context of each contiguous insertions and deletions between revision r_i and r_{i+1} we introduce two bitmaps (Figures 7 and 8). Both bitmaps are of the size equal to $string\_tokens\_i$ in r_i. The first bitmap $deleted\_mask_i$ corresponds to $deletion_i$, where 1 indicates the token positions which will be removed in $e_{<i,i+1>}$ (otherwise, a 0 is used). The second bitmap $inserted\_mask_i$ corresponds to $insertion_{i+1}$, where 1 indicates the tokens that will

Revision Id 0

Content	st@rt	A	tree	was	standing	Close	!	&end
TokTrack id	-1	10	11	12	13	14	15	-2
Inserted Mask								
Deleted Mask	0	0	0	0	0	0	0	0
1d diff	0	0	0	0	0	0	0	

Revision Id 1

Content	st@rt	A	tree	was	standing	close	!	For	real	.	&end
TokTrack id	-1	10	11	12	13	14	15	16	17	18	-2
Inserted Mask	0	0	0	0	0	0	0	1	1	1	0
1d difff	0	0	0	0	0	0	1	0	0	-1	0
Deleted Mask											

Figure 7: Edit $e_{<0,1>}$

be inserted in $e_{<i,i+1>}$ (otherwise, a 0 is used). Since each WikiWho token is unique, we do not need to worry about ambiguity of tokens with the same text, and setting up this mask runs in $O(n)$.
Next, we take the difference of each element in both masks with its next value to mark the $left\_context\_start$ and $right\_context\_start$ (1d diff in Figures 7 and 8). If there is a 1, then a $left\_context\_start$ has been detected, whereas if there is a -1, then the $right\_context\_start$ is located in the subsequent position, i.e. the one after -1. All the tokens which come between $left\_context\_start$ and $right\_context\_start$ are contiguous insertions or deletions with a shared context, i.e. a change object. Once we are able to identify all pairs of 1 and -1 we identify all contiguous insertions and deletions along with its context, and therefore all the change objects.
The algorithm runs in O(n) time which is just the search of 1 and -1 in mask difference.

Figure 7 shows the Edit $e_{<0,1>}$ including the special token added in start and end to mark the start and end of revision. Difference of consecutive element in inserted_mask is 1d diff. As stated above the tokens “For”, “real” and “.” were added in revision r_1 between "!" and especial end token "end". Position where there is a 1 in 1d diff of mask gives us the first token of left context ($left_context_token$) i.e “!”, and positions after a -1 in 1d diff gives us the first token in the right context

Revision Id 1

	A	B	C	D	E	F	G	H	I	J	K	L
1	Content	st@rt	A	tree	was	standing	close	!	For	real	.	&end
2	TokTrack id	-1	10	11	12	13	14	15	16	17	18	-2
3	Inserted Mask											
4	Deleted Mask	0	0	0	0	0	1	1	0	1	0	0
5	1d Diff	0	0	0	0	1	0	-1	1	-1	0	

Revision Id 2

	A	B	C	D	E	F	G	H	I	J	K	L
1	Content	st@rt	A	tree	was	standing	nearby	.	For	sure	.	&end
2	TokTrack id	-1	10	11	12	13	19	20	16	21	18	-2
3	Inserted Mask	0	0	0	0	0	1	1	0	1	0	0
4	1d diff	0	0	0	0	1	0	-1	1	-1	0	
5	Deleted Mask											

Figure 8: Edit $e_{<1,2>}$

($right_context_token$), i.e. especial end character. Our contiguous inserted token is located between the first token in left context and first token in right context. In the example above contiguous insertions is 'For', "real", "." located between the markers 1 (exclusive) and -1 (inclusive). Our algorithm 1 is able to identify insertions at the end because of the especial token marking the end of the content in r_1. Similarly, contiguous insertions at the start of revision can be identified too. Thus, our algorithm 1 is efficiently able to identify all contiguous insertions in a revision along with its position as well as their respective $left\_context\_start$ and $right\_context\_start$ tokens.
Similarly, figure 8 shows contiguous deleted tokens in r_1. We have a deleted mask bitmap corresponding to each token in r_1. If we take the difference of consecutive values of bitmap, we can localise two contiguous deletes in r_2, indicated by markers 1 and -1. So, two contiguous deleted token units are identified, ("close", "!" and "real"), with their respective $left\_context\_start$ and $right\_context\_start$ tokens as (standing, For) and (for, .) respectively.

4.1.3 Merging of contiguous edits

Once we have identified contiguous insertions and deletions in each revision r_i which are fine grained in the $wiki\_who\_tokens_i \in r_i$ by their respective $left\_context\_start$ and $right\_context\_start$ tokens context, we need to identify fine grained gap of contiguous insertions and deletions in each edit $e_{<i,+1>}$, i.e. change objects. For identifying contiguous deletions, we divide $deletions_i \in e_{<i,+1>}$ in units of tokens which are present at contiguous positions in the r_i, but get removed in the next revision r_{i+1}. Similarly, for identifying contiguous insertions, we divide $insertions_{i+1} \in e_{<i,+1>}$ into units of tokens inserted contiguously between r_i to get r_{i+1}. Both of these divisions are marked by $left\_context\_start$ and $right\_context\_start$ tokens, so in order to identify the gaps of inserted and deleted tokens in $e_{<i,+1>}$ we associated these two list of tokens. As both the $left\_context\_start$ and $right\_context\_start$ tokens do not change in $e_{<i,+1>}$, we associate all the contiguous insertions and deletions which share $left\_context\_start$ and $right\_context\_start$ in $e_{<i,+1>}$. For example, in Figure 8, contiguous deletions of "close", "!" in R_1 are associated with contiguous insertions of "nearby", "." as their $left\_context\_start$ and $right\_context\_start$ is same. $left\_context\_start$ is "Standing" and $right\_context\_start$ is "for". These two associated units are called the gap of the $change\_object$.
If no association exists for an insertion or deletion unit, then an empty list is assigned in place of that missing counterpart. This indicates that between $left\_context\_start$ and $right\_context\_start$ of unmatched changed object, only tokens were inserted or deleted in $e_{<i,+1>}$. Unlike Bykau et al. (2015), which only captures gaps where both insertions and deletions tokens are present, our merge process finds the change object gap where one of insertions and deletions can be empty. Our algorithm thus finds all the token in Bykau et al. (2015) as well as also the ones where insertions or deletions are empty.

To summarise the change object identification algorithm (1), we identify change objects in edit history of an article by processing the set of revisions in time ordered manner. We start from r_1 until r_n and identify tokens added by editors in each $e_{<i,i+1>}$ between consecutive revisions r_i and r_{i+1}. We model the edition process as an editor $editor_i$ inspecting the existing revision r_i and further adding and removing string tokens at certain positions in existing $string\_tokens_i$ to create a new revision r_{i+1} with $string\_tokens_{i+1}$. This edit process gives us $e_{<i,i+1>}$, consisting of a set of inserted tokens $insertion_{i+1}$ and a set of deleted tokens $deletion_i$. We divide these inserted and deleted tokens which are at contiguous position in r_i into contiguous insertions and deletions. We further associate these contiguous insertions and deletions into one change object if they share $left\_context\_start$ and $right\_context\_start$. Each change object contains added and removed tokens at the contiguous positions between a fixed $left\_context\_start$ and a fixed $right\_context\_start$. Change objects built by this association consist of two ordered list of tokens, both of which are at contiguous positions in edit $e_{<i,i+1>}$ and between a fixed context starting at $left\_context\_start$ and $right\_context\_start$. The first list contains the tokens which are present at continuous positions in the previous revisions $r_i \in edit< i, i+1 >$, but get removed in the next revision r_{i+1}. The second list contains the tokens added contiguously in r_{i+1} at the same position from where the tokens in the first list are removed in the previous revision r_i.
In this way, each edit $e_{<i,i+1>}$ is divided into a list of change_object consisting of gaps of inserted and deleted tokens with a left and right context. Once we run our algorithm 1 for all edits $e_{<i,i+1>}$ in revision history of an article, we get the change object corresponding to complete revision history of an article. Figure **??** shows the change object identified by our algorithm 1 for list of edits $e_{<0,1>}, e_{<1,2>}$.
Make $Inserted_{i+1}$ with the size of $Tokens_i$ where each position, which has new tokens, will be set to 1 and remaining to zero. Take difference between consecutive element of $\vec{Deleted}_i$ and $\vec{Inserted}_{i+1}$ to find contiguous insertion and deleted positions. As $\vec{Deleted}_i$ and $\vec{Inserted}_{i+1}$ contains 1 at insertion and deletion positions, difference vector will contain -1 at left neighbour of contiguous edited positions, 1 at end of contiguous edited positions and 0 in the remaining positions. In the difference vector of $\vec{Deleted}_i$ and $\vec{Inserted}_{i+1}$, -1 denotes left neighbour position where contiguous insertion or deletion started and 1 denotes the position where contiguous insertion and deletion stopped. One position ahead where 1 is coming in difference vector is right neighbour.
Merge the deleted and inserted token from $\vec{Deleted}_i$ and $\vec{Inserted}_{i+1}$ which share the same left and right neighbour. This gives us the change objects between the right and the left neighbour.

4.2 Change Vectors

Before we cluster change objects, we encode each of them in a fixed dimensional vector space using vector space modeling on its left and right context and these vec-

Algorithm 1 Change Object Identification

Require: $revision\_list$ ▷ List of all revisions

1: **for** $i \leftarrow range(len(revision\_list) - 1)$ **do**
2: Pad $tok\_track\_id_i$ and $tok\_track\_id_{i+1}$ at start and end with special tokenId to mark start and end positions.
3: $deleted\_1d\_diff_i \leftarrow deleted\_mask_i[1:] - deleted\_mask_i[:-1]$
▷ $deleted\_mask_i[j] - deleted\_mask_i[j+1]$
4: $inserted\_1d\_diff_{i+1} \leftarrow inserted\_mask_{i+1}[1:] - inserted\_mask_{i+1}[:-1]$
▷ $inserted\_mask_{i+1}[j] - inserted\_mask_{i+1}[j+1]$
5: $deletion\_start\_pos_i \leftarrow deleted\_1d\_diff_i == -1$
6: $deletion\_end\_pos_i \leftarrow deleted\_1d\_diff_i == 1$
7: $insertion\_start\_pos_{i+1} \leftarrow inserted\_1d\_diff_{i+1} == -1$
8: $insertion\_end\_pos_{i+1} \leftarrow inserted\_1d\_diff_{i+1} == 1$
9: $deleted\_positions\_matrix_i \leftarrow concat(deletion\_start\_pos_l; m, i, deletion\_end\_pos_i)$
10: $inserted\_positions\_matrix_{i+1} \leftarrow concat(insertion\_start\_pos_{i+1}, insertion\_end\_pos_{i+1})$
11: $change\_matrix_{<i,i+1>} \leftarrow outer\_join([deleted\_positions\_matrix_i, inserted\_positions\_matrix_{i+1}], c$
$[deletion\_start\_pos, insertion\_start\_pos])$
12: **end for**

tors corresponding to change objects are called change vectors.
Each of our change object gap is bounded by $left\_context\_start$ and $right\_context\_start$ tokens, which we have already defined to be first token which was not edited in both right and left sides of contiguous edits in an edit $e_{<i,i+1>}$. Each gap in an edit $e_{<i,i+1>}$ comes between two revisions, but we have fixed both of its right and left context to be in revision r_i, as this is the context that editor of r_{i+1} sees before creating r_i Bykau et al. (2015). Therefore, left and right contexts of each change object are in r_i and start at $left\_context\_start$ and $right\_context\_start$ tokens which are used to mark contiguous insertions and deletions and merge them. In order to mark the end of context, we introduce a parameter to our model $context\_length$, which controls the number of tokens used on the left side of $left\_context\_start$ and the right side of $right\_context\_start$ to create change vectors. Note that $context\_length$ is the only one parameter which limits the size of both context, meaning that the same amount of tokens will be used on both sides of the change object to create change vectors. We pick tokens from start of left and right context to the $context\_length$ and make two list of tokens, namely left context tokens and right context tokens.
Let $pos(token)$ be the position of token in $string\_tokens_i$, and let $len(string\_tokens_i)$ be the total length of $string\_tokens_i$. Then, we define left context tokens $left\_context\_tokens$ of a change object coming in $e_{<i,i+1>}$ as the list of tokens in who are at contiguous positions in $string\_tokens_i \in R_i$, and that are located between $pos(left\_context\_start) - context\_length$ to $pos(left\_context\_start)$.
If $pos(left\_context\_start) < context\_length$ then we take the contiguous tokens between 0 to $pos(left\_context\_start)$.

Similarly, we define right context tokens $right\_context\_tokens$ of a change object coming in $e_{<i,i+1>}$ as list of tokens in which are at contiguous positions in $string\_tokens_i \in R_i$, and that are located between $pos(right\_context\_start)$ to $pos(right\_context\_start) context\_length$. If $pos(len(string\_tokens_i) > right\_context\_start) + context\_length$ then we take the contiguous tokens between $pos(right\_context\_start)$ to $len(string\_tokens_i)$.
For efficient representation of string tokens, we use Word vectors, given by Mikolov et al., which semantically represents word vectors into a fixed dimensional representation. Using Word vectors, we encode our left and right context into a fixed dimension space. As discussed in background section, these word vectors are trained on Wikipedia corpus and gives R^{300} vector corresponding to each word in its corpus vocabulary. Owing to finite RAM size and for efficient computation, we keep vocabulary size for word embedding to be 1 million words sorted by their rank of occurrence in the training corpus. We use well known Řehůřek and Sojka (2010) Gensim Library to load the vectors corresponding to top 1 million words sorted by their rank according to frequency of occurrence in the corpus. In top 1 million vocabulary size, we perform two kinds of cleaning, first one is to drop the top 10 ranked words from vocabulary and the second is to drop any word with the length of size less than or equal to 3. We remove top 10 ranked words because top ranked words are the most frequent words in corpus and as discussed in Tf-Idf in Section (3) more frequent words carry less meaning in corpus and are generally generic words carrying very less meaning. We remove words with size less than 3 because most of the words of size less than 3 are pronouns or symbols of less significance.
Once we have cleaned the vocabulary of our embedding vectors, we define a mapping function $embedding_map(word)$. This mapping function takes each word in our cleaned vocabulary and return its corresponding word vectors of R^{300} from Mikolov et al.. If the word is out of vocabulary, it has a zero vector in R^{300}. Mapping function $embedding\_map(word)$ helps us to ignore all the other words which are not in cleaned vocabulary $cleaned\_vocab$, i.e. the words which are removed in cleaning or do not exist in 1 million vocabulary set.
Let $word\_vector(word)$ be a function in Řehůřek and Sojka (2010) Gensim which maps each word to R^{300} embedding vector, and let $cleaned\_vocab$ be a set of cleaned vocabulary obtained above, then $embedding\_map(word)$ is defined as:

$$embedding\_map(word) = \begin{cases} word\_vector(word) \in R^{300} & \text{if } word \in cleaned\_vocab \\ 0 \in R^{300} & \text{if } word \notin cleaned\_vocab \end{cases} \tag{18}$$

We use the cleaned word embedding $cleaned\_vocab$ and $embedding\_map(word)$ to average tokens in our left and right context into a vector of R^{300}. We finally concatenate both right and left context vector of R^300 to get R^600 change vector. Formally,

Edit	Left Context	Right Context
<0,1>	R^{300}	R^{300}
<1,2>	R^{300}	R^{300}
<1,2>	R^{300}	R^{300}

Figure 9: Change Vectors

$$\textbf{Left\_Context\_Vector} = \frac{1}{len(left\_context)} * \sum_{i=0}^{len(left\_context)} embedding\_map(left\_context[i]) \tag{19}$$

$$\textbf{Right\_Context\_Vector} = \frac{1}{len(right\_context)} * \sum_{i=0}^{len(right\_context)} embedding\_map(right\_context[i]) \tag{20}$$

$$\textbf{Change\_Vector} = \Big[\textbf{Left\_Context\_Vector}; \textbf{Right\_Context\_Vector}\Big] \tag{21}$$

As an example, Figure 9 shows the list of change vectors that corresponds to the change object list in **??**.

As the semantic meaning of a change object lies in the token in its $left\_context$ and $right\_context$, we hypothesize that the change vectors created from left and right context surrounding the gap of change object contain information that is able to associate insertions and deletions in the gap of a change object. One important parameter which we consider in encoding semantics from left and right context is $context\_length$, i.e. the number of tokens included in the left and the right context, because it has effects on capturing the meaning of inserted and deleted token in the gap as more or less tokens are considered. Different values of $context\_length$ allow us to capture different number of tokens before and after gap of change object. As described earlier, for each change object coming in edit $e_{<i,i+1>}$ we consider its context in r_i, so by varying the length of the context we pick varying amount of tokens in left and right neighbour of gap in r_i. For each value of $context\_length$ we get a corresponding change vector. We, therefore, create change vector matrix for set of change objects corresponding to $Edit\_history$ of an article. Each row in change vector matrix corresponds to a change object. For each values of context length we

get a corresponding change vector matrix.

4.3 Grouping and Evaluation

We utilise Density-based spatial clustering of applications with noise(DBSCAN) from Ester et al. (1996) to cluster $\mathbf{Change\_Vector} \in R^{600}$ corresponding to each value of $context\_length$. In order to cluster change vectors, DBSCAN needs to calculate distance between change vectors. We use euclidean distance metric to calculate distances between each pair of change vectors. We use euclidean distance as distance metric because it satisfies all the three criteria of being a metric as described in Clustering section of Section (3). Most of these change vectors are about change object which are general edits and do not belong to any groups. We hypothesize that the groups of change object which are about similar subject matter lie closer in distance matrix to other change vectors, thus forming areas of high density. Change object which are evenly distributed in distance matrix should be about general edit tokens, not signifying any specific edit topic, thus forming low density region of not particular interest. Being a density based clustering algorithm, DBSCAN is able to identify high density groups and leave out low density regions. To identify high density groups and leave all other points, DBSCAN has two parameters, minimum sample size and epsilon. As explained in Section (3), DBSCAN uses these two parameters to create core group of points and keep adding new points in core groups to identify high density regions.
We group Change Objects by clustering R^{600} dimensional Change Vectors corresponding to each change object and $context\_length$. We, therefore, have three parameters for our grouping algorithm: $context\_length$ is the first parameter which gives us change vectors to cluster and remaining two parameters are $min\_samples$ and eps, which are the parameters of DBSCAN algorithm. To evaluate the clusters created by DBSCAN algorithm, we run it on various combinations of these three parameters. We utilise DBSCAN algorithm from Pedregosa et al. (2011) library for generating clusters labels for various parameters of epsilon distance, eps and minimum number of samples, $min\_samples$ for each change vector corresponding to $context\_length$. As DBSCAN clusters are unsupervised labels, we first evaluate them instrincally on goodness of labels. After intrinsic evaluation, we do extrinsic evaluation of the algorithm against golden dataset.

4.3.1 Intrinsic Evaluation

DBSCAN cluster gives clusters of uneven sizes depending upon the parameters eps and $min\_samples$. We analyse this unevenness as the first parameter for intrinsic evaluation of DBSCAN clusters. We calculate various descriptive statistics on the clusters sizes of DBSCAN for analysis. We calculate the Gini co-efficient of the size of the clusters created by a particular set of parameter too. Lower values of Gini suggest more even sized clusters identified by DBSCAN.

A stable cluster of change objects returned by DBSCAN should be from similar positions in its revision content. In order to quantify the position of change object in content of its revision, we introduce the concept of relative position of change object. Relative position of a change object belonging to an edit $e_{<i,i+1>}$ is defined as the position of first token in left context in $wiki\_who\_string_tokens_i$ in r_i divided by length of total tokens in $wiki\_who\_string\_tokens_i$ inr_i. Cluster of change object is more stable if it contains lower Shanon's entropy Shanon (1948) of relative positions. Relative position is a ratio variable between 0 and 1. In order to calculate kinds of relative positions, we round it off to two decimal places. Rounding off gives us finite categories of relative positions for all change objects, making it a countable categorical variable. In order to quantify kinds of different relative position categories present in a cluster, we use Shanon's entropy Shanon (1948), which help us quantify randomness of relative position category in a cluster. Lower randomness in relative position entropy of a cluster signifies it contains change object with few kinds of relative position. It can be interpreted as the cluster having change objects with less probability of coming from random places in their respective revisions. In order to calculate entropy of relative positions of a cluster generated by our algorithm, we calculate the frequency of rounded off relative positions in the cluster. We pass this generated frequency to the entropy function using Scipy library from Pasternack and Roth (2008). This gives us entropy of relative position of each cluster generated by our algorithm. We further propose an aggregated metric, which quantifies the stability of clustering algorithm with respect to relative position of change object contained in the clusters, by taking weighted average of relative position of entropy of each cluster with its size. Reason for weighting the entropy before taking its average is that the smaller cluster size will tend to have lower entropy. As the cluster size is small, chances of getting change object from random places in revision decreases. So, weighted average favours bigger clusters with less randomness in them.

We have created change vector by projecting left and right context of change object. We hypothesise that clusters created by these change object will contain gap about similar items. So a cluster of change object is more likely to have gaps related to the same topic, if the gap contains same kind of string tokens. We define weighted token entropy of string tokens in gap similar to weighted relative position entropy to quantify randomness in gap of change object in clusters created by DBSCAN. For purpose of weighted token entropy of string tokens in gap we first define token entropy of string tokens in gap of change objects of each cluster. Token entropy of a single cluster is Shanon's entropy of frequency of total string tokens in gap of all change object in a cluster.

Token entropy measures the randomness in gap of change object in a cluster because of presence of different kind of string tokens. So a cluster will have lower token entropy if it has fewer kind of string tokens, hence signifying that cluster on change vector is able to create group of gap which are about same topic. Similar to weighted relative position entropy we take weighted average of token entropy of each cluster

with its cluster size to negate for the effect of smaller cluster having lesser entropy in their gap.
In order to identify the cluster stability or different values of $context\_length$, eps and $min\_samples$ we calculate change object of 17 articles shown in Table 10

After calculating change object for each article, we make two change vectors corresponding to context length of 4 and 10.
We, further, fix the $min\_samples$ as 5 and run DBSCAN for eps value in range of 0.5 to 4 with step size of 0.5. We calculate intrinsic evaluation metric and analyse the best parameter of context length and eps for all the articles.

We also re-implement Bykau et al. (2015) for their suggested parameter for all 17 articles shown in table 10 and compare it with our cluster using Fowlkes Mallows score E.B. Fowlkes (1983).

4.3.2 Extrinsic Evaluation

We select the John Loggie Baird Wikipedia page to create a gold standard dataset of insertions and deletions associated (or not) with his nationality. Wikipedia contributors (2019a) We select the nationality of John Loggie Baird as it appears in the list of Wikipedia Lamest Edit Wars and we consider its size appropriate for our evaluation work. The page size is around 32KB (2.5K words as of 18.02.2019), which, according to Wikipedia size guideline Wikipedia contributors (2019b) is neither too short (< 1KB, or stub), nor one that justify division of the page by its length alone (< 40KB).
We use the WikiWho API [1] and the wikiwho_wrapper Python library [2] to identify the tokens that represent the main nationality conflict (i.e. British vs Scottish). Upon manually checking the context in which these tokens appear, we extend the dataset to other tokens related to places that could have also been related or confounded to nationality (i.e. Scotland, United, Kingdom, UK, Englain, England, London, Helensburgh, Bute, Dunbartonshire, Argyll). We consider that our procedure gives an almost full coverage of the editions that are related to of the nationality Edit War.
The dataset is manually annotated indicating if each token is or not related to nationality. We also annotate whether the token (1) is related to the birth place (which context could be easily confounded with nationality), (2) is part of a Wikipedia link (which context often comprised special Wikipedia markup characters which, we presume, could make identification difficult), or (3) is part of a large group of tokens inserted or deleted in simultaneously in the same position. Table in figure 11 summarizes the annotated categories and the number of cases found in the dataset:

We create ground truth from this annotated category of inserted and deleted tokens where Nationality is true and it's not in Bulk, against which we can compare the groups found by our clustering algorithm.

[1] https://api.wikiwho.net/
[2] https://pypi.org/project/wikiwho-wrapper/

Article name	No of change object
Crusades	45523
Friedrich_Nietzsche	41301
Freddie_Mercury	40276
Truth	31777
IQ_and_the_Wealth_of_Nations	21173
Berlin_Wall	19641
E_(mathematical_constant)	14311
Yugoslavia	11960
Censorship	10987
Mama's_Family	10887
Human_cloning	10529
Systemic_lupus_erythematosus	10032
Solar_power	7595
Gambling	5999
Electronic_voting	5544
John Loggie Baird	4900
Violence_against_Muslims_in_India	3933

Figure 10: List of Wikipedia article with sizes

Category	Count	Description
Nationality	497	Does the token relate to nationality?
Birth place	394	Does the token relate to place of birth?
Link	554	Does the token belongs to a Wikipedia internal link?
Bulk	1029	Is the token part of a large group of other tokens inserted or removed simultaneously?
Total	1775	

Figure 11: Distribution of annotated cases. The first column presents the annotated category in the dataset, the second column the number of positive cases annotated with that category, and the third column a detailed description of the category. The categories Nationality and Birth Place are mutually exclusive, but the categories Link and Bulk can overlap with any of the others. The total in the last row at the end is therefore not the sum of the individual counts.

In order to do extrinsic evaluation of our algorithm, we need to create cluster of each token in our golden data using our algorithm with different values of $context\_length$, eps and $min\_samples$. We use V-measure from Rosenberg and Hirschberg (2007) to evaluate extrinsic clusters. V-measure is harmonic mean of Completeness and Homogeneity measure. These three measures are analogous to precision, recall and f-score from classification algorithms. We evaluate our clustering algorithm on all three parameters and find optimal values of $context\_length$, eps and $min\_samples$. For comparing performance of our model with Bykau et al. (2015) we run their model on our change object with all the parameters of $context\_length$, eps and $min\_samples$. We compare and contrast Completeness, homogeneity and V-measure scores of Bykau et al. (2015) on different combinations of three parameters.
To give a fair comparison, we also reimplement Bykau et al. (2015) by dropping three kinds of change object from our model to match with change object of Bykau et al. (2015). As change object of Bykau et al. (2015) has only those gap where both insertion and deletion are present, we drop all the change object with only insertions or deletions. Next we remove all the change object whose length of inserted or deleted token is more than 5 following suggestion from Bykau et al. (2015). Lastly we drop all the change objects whose gap is not repeated by two users, minimum user support condition of Bykau et al. (2015). We run their change object group identification algorithm with same parameters of $context\_samples$, eps and $min\_samples$.
We run all three models; ours, Bykau et al. (2015) with our change object, Bykau et al. (2015) with reduced change object on various values of $context\_length$, eps and $min\_samples$. We fix $min\_samples$ as 2, $context\_length$ as 4,6,8,10,15,30 and we varied eps from 0.5 to 4 with step size of 0.5. We compare and contrast results for all the values of the parameters in the results section.

5 Results

We have done all the implementation of experiments regarding identifying fine grained groups in change object in Python Programming language using packages in Scipy stack Jones et al. (2001–). All steps of the implementation, as represented in Architecture diagram of 2 are done in reproducible IPython notebooks Pérez and Granger (2007). Change object detection algorithm is written entirely in Pandas McKinney (2010) and Numpy Oliphant (2015) libraries. Clustering and evaluation is done using Scikit-learn machine learning libraryPedregosa et al. (2011). In the following section, we present the result of our analysis in detail, starting from results on stability of DBSCAN clusters using intrinsic evaluation, followed by Fowlkes–Mallows index agreement of our clustering results with Bykau et al. (2015) and at last we present results of extrinsic evaluation with class labels from annotated Golden Dataset using V-Measure Rosenberg and Hirschberg (2007).

5.1 Intrinsic evaluation

To measure the stability of the cluster, we do intrinsic evaluation of various parameters of $context\_length$, eps and $min\_samples$ on all 17 articles in Table 10. Parameter of evaluation is shown in Table 12. We first analyse the No of cluster identified vs outliers left by DBSCAN to analyse the effect on eps in leaving out the noise. We observe that absolute value of cluster size is directly correlated to change object size as can be observed from Graph in figures 13 and 14. When we normalize change object size with maximum size of change object across all eps for a given article, we observe that for all the article change object and outliers follow same pattern in both 13 and 14. This suggests that both the $context\_lenght$ have similar variability with number of cluster and cluster size normalised by maximum no. of clusters following similar distribution across all the articles. Variation of outliers with eps is a strictly decreasing function implying that as the eps is increased more change objects, whose contexts are different, are added in the group of cluster by DBSCAN. Number of clusters are observed to have a maxima for value ofeps in [1.5,2.0] then decreases to 1. This pattern implies that when value of eps goes more than 1.5, clusters start to merge together into one cluster.

In order to further investigate the effect of cluster sizes on the stability of clusters. We compare Gini coefficient of cluster size distribution, mean cluster size, max cluster size, standard deviation of cluster size, gap entropy and position entropy and we plot the respective results in 16 and 15. We observe that values of all the stability measure show a low and stable value before 1.5. After which it drastically increases. For $context\_length$ of 10 we observe that for most articles Gini has minima between 1 and 1.5, while gap entropy and position entropy is also lower at 1.0. Hence we choose value of 1.0 for eps to further continue our intrinsic value analysis.

Parameter name	Range
Context_length	{4,10}
min_samples	5
eps	{0.25,0.5,0.75, … , 3.25,3.75,4.0}

Figure 12: Table shows the Values of parameters for intrinsic evaluation. Using the Combination of these parameters we analyse intrinsic stability of clusters.

5.2 Agreement with Bykau et al. (2015)clusters

We compare our clusters created on parameters of $context\_length = 4, 10$, $min\_samples = 5$ and $eps = 1$ with clusters created using Bykau et al. (2015) with all the parameters suggested by them. Figure 18 shows a bar plot representing the agreement of our clusters with Bykau et al. (2015) clusters for values of $context\_length$ 0 and 1 using Fowlkes–Mallows index.
We observe clusters corresponding to both $context\_length$ to have high value of Fowlkes–Mallows index, but index value for $context\_length$ 4 is lower than 10 for all articles. Upon observing Box Plot corresponding to both values of $context\_length$, we see that $context\_length = 10$ has more spread between quartile but it has only one outlier while $context\_length = 4$ has two outliers. Values of outliers along with spread of box plot suggest that for $context\_length = 4$,most of the articles have similar agreement but agreement of two articles is very low. Whereas for $context\_length = 10$, we observe that values have higher spread but spread is towards higher agreement. The $context\_lenght = 10$ has higher Fowlkes–Mallows index agreement as well as only one low outlier value. These low value of outliers need further analysis to understand the reason behind low agreement index.

5.3 Extrinsic evaluation

We run three related metric of Completeness, Homogeneity and V-measure to validate our results and compare it with Bykau et al. (2015). We run analysis on all three clustering algorithm we have; our change object detection algorithm, Bykau et al.

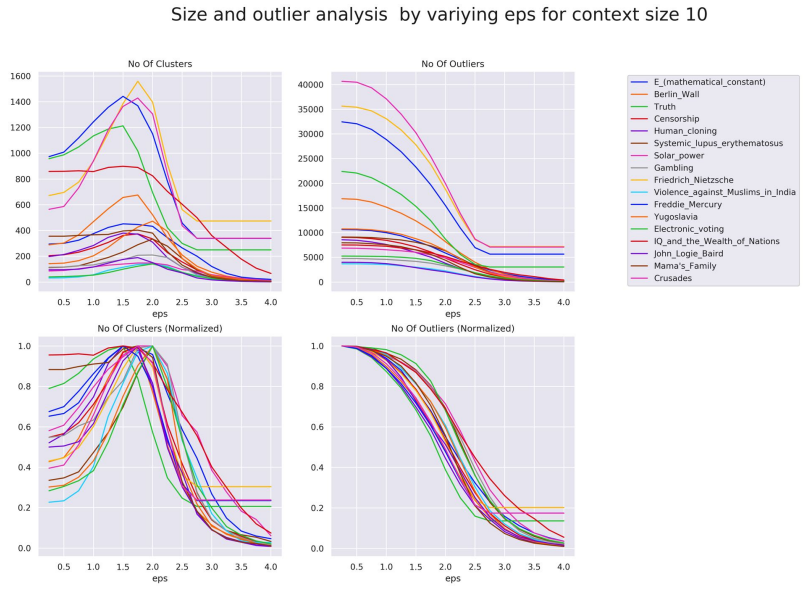


Figure 13: Size vs Outliers analysis for $context\_length$ of 10. X-axis shows the eps parameter of DBSCAN while y axis shows the number of points in clusters.First row shows no of cluster and no of cluster normalised by maximum number of clusters. Second row shows no of outliers and its normalised counterpart. No of cluster plot shows that it peaks at value of eps around 1 and 2. Outliers plots shows that as the value of eps from 1 to 4, no of outliers tends to go to zero.

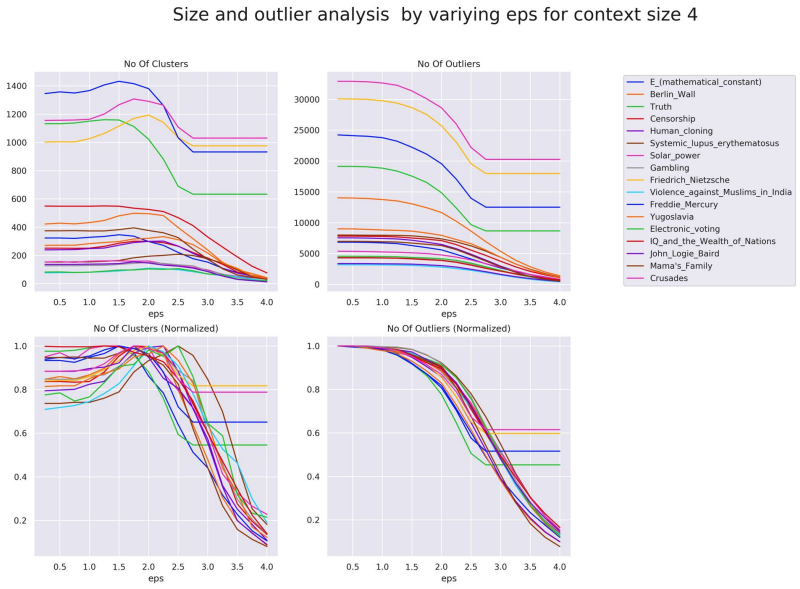


Figure 14: Size vs Outliers analysis for $context\_length$ of 4. X-axis shows the eps parameter of DBSCAN while y axis shows the number of points in clusters. First row shows no of cluster and no of cluster normalised by maximum number of clusters. Second row shows no of outliers and its normalised counterpart. No of cluster plot shows that it peaks at value of eps around 1 and 2. Outliers plots shows that as the value of eps from 1 to 4, no of outliers tends to go to zero

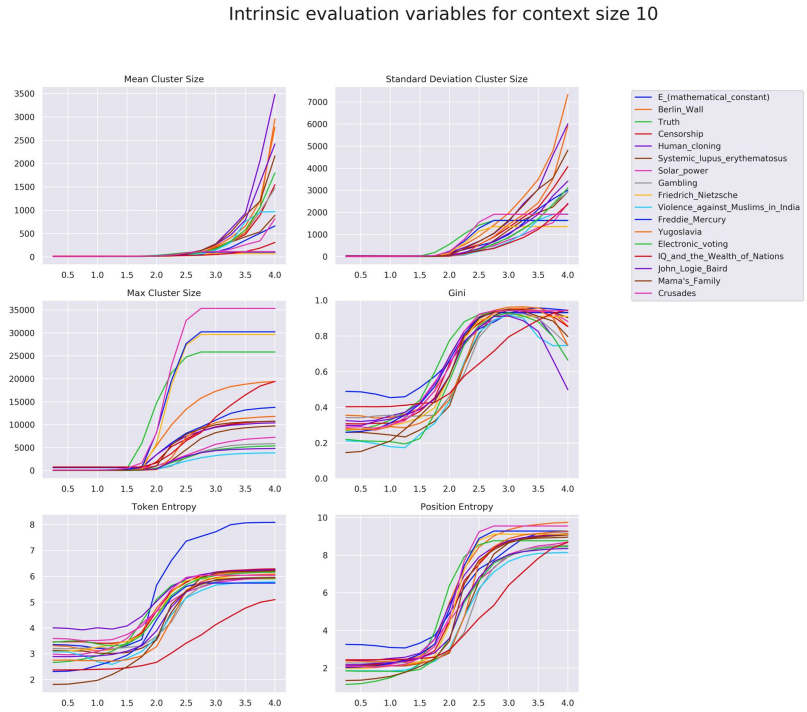


Figure 15: Intrinsic evaluation for $context\_length$ of 10. X-axis shows the value of eps, while Y axis shows the value of all the intrinsic evaluation parameter.

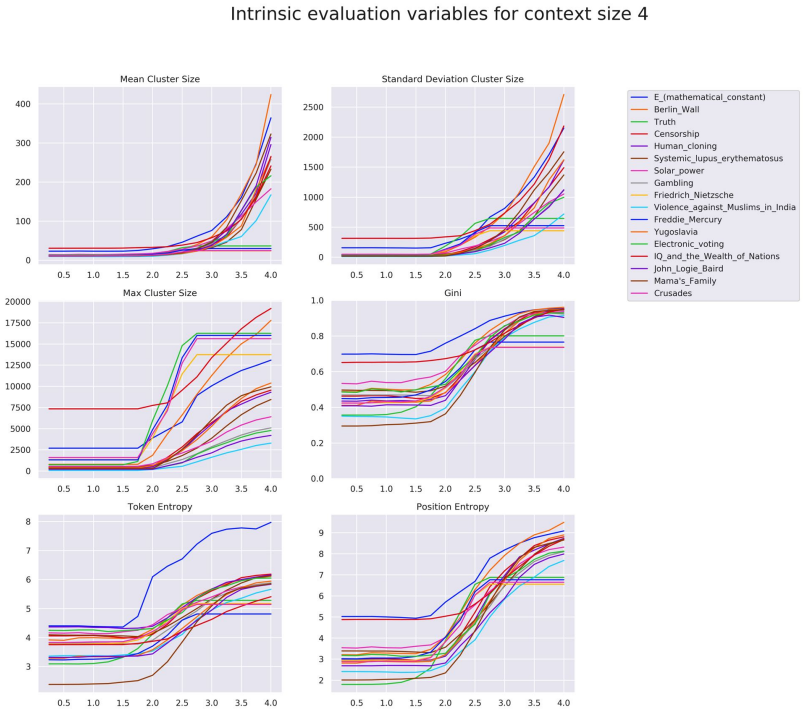
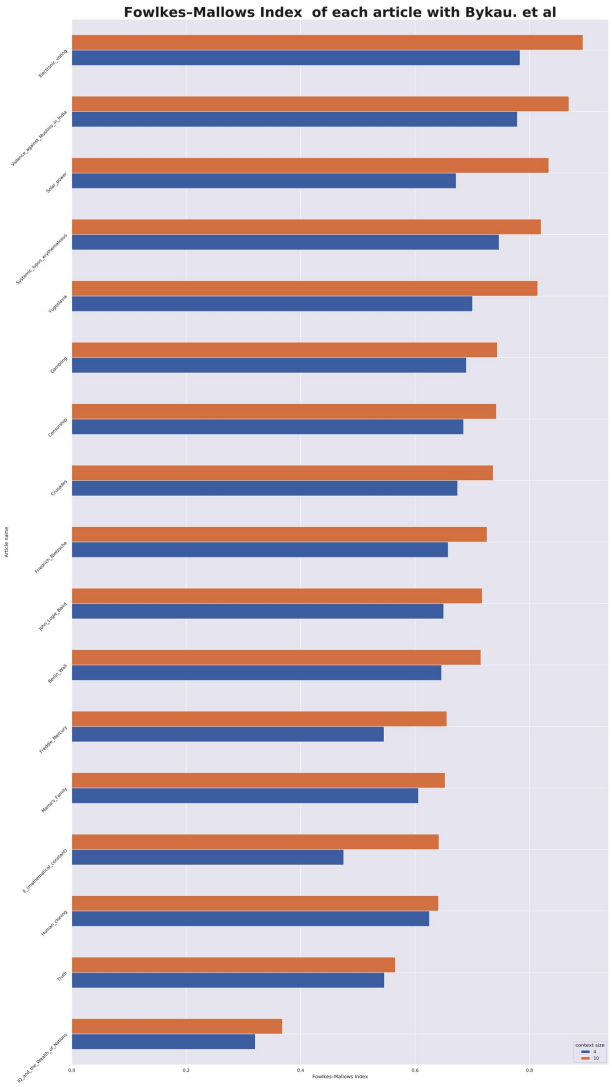


Figure 16: Intrinsic evaluation for $context\_length$ of 4. X-axis shows the value of eps, while Y axis shows the value of all the intrinsic evaluation parameter.



: Figure 17
Fowlkes–Mallows Index between our clusters and Bykau et al. (2015). X-Axis shows the Index while Y-axis has name of article. IQ$and_the_wealth_of_nations has the lowest agreement.$

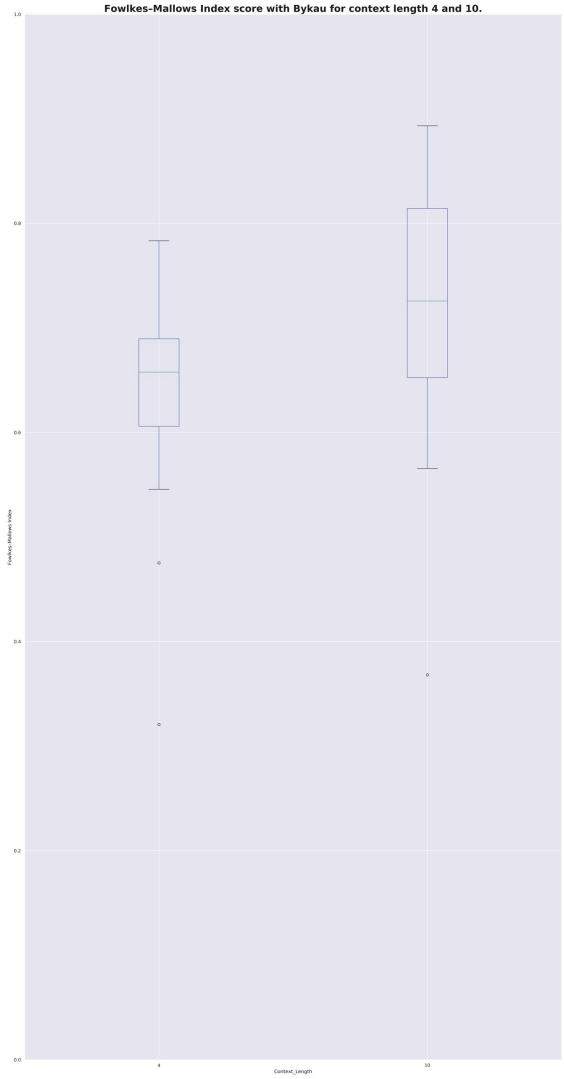


Figure 18: Fowlkes–Mallows Index between our clusters and Bykau et al. (2015) (context length effect). Two outliers in case of $context\_length = 4$ and one outliers in case of $context\_length = 10$

Parameter name	Range
Context_length	{2,4,8,10, 15, 20,25,30}
min_samples	{2,5,50}
eps	{0.25,0.5,0.75, … , 3.25,3.75,4.0}

Figure 19: Table shows the Values of parameters for extrinsic evaluation. Combination of these parameters were used to evaluate clusters against Golden dataset.

(2015) with optimisation and without optimisation.
As shown in Table 12, for intrinsic evaluation discussed previously we fix the value of $min\_samples = 5$ and $context\_length = [4, 10]$ and analyse stability of cluster on size, no. of outlier, gap entropy, position entropy etc. For extrinsic evaluation of our data as shown in 19, we vary the values of $context\_length$ from 2 to 30 to study effect of $context\_length$ in all the three clustering algorithms.

5.3.1 Optimal value of parameters

Having chosen the range of $context\_length$, we first validate our clustering algorithm on the $min\_samples = 5$ to analyse the value of homogenity and completeness. Our golden data set has annotation only for the presence of class of "nationality" and we leave all the other annotations as not known. Thus, in Figure 20, we observe that the value of homogenity is always higher compared to completeness for all values of $context\_length$.
We know that homogeneity and completeness are competing measure, so in order to analyse the bounding value of completeness we increase the value of $min\_samples = 50$, hoping to get cluster, big enough to have high values of completeness and low values of homogenity. In Figure 21, we observe that for smaller values of eps, completeness goes to one but homogenity becomes zero thus resulting in very low value of V-measure. Hence this suggests that high values of eps, although give high Completeness in clusters, but is not optimal for Homogeneity and Vmeasure. We further

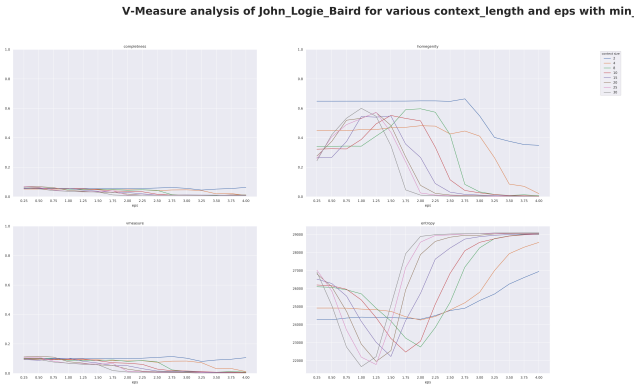


Figure 20: Analysis of extrinsic measures of Completeness, homogenity, V-measure with intrinsic measure of gap entropy for $min\_samples = 5$. X-axis represents different values of eps and y-axis represents the value of the evaluation metrics. Negative correlation in shape of gap entropy with extrinsic evaluation measures is observed for all values of $context\_length$

analyse relation of cluster size and Completeness. We observe that we get the cluster size of 7 for low value of Completeness and when the Completeness value becomes 1 all the change object gets clusters in one cluster of outlier.
Finally in order to maximise for homogenity and V-measure we decrease the value of $min\_samples$ to its lowest possible value of 2. We observe, as plotted in Figure 22, value of homogeneity for all the context length increases to about 0.7. Value of entropy of change object in gap also decreases considerable, hence we pick the value of $min\_samples$ to continue comparing our algorithm with both the implementation of Bykau et al. (2015).

5.3.2 Comparison with Bykau et al. (2015)

Plot in figure 24 shows results of V-measure analysis of jaccard clustering of neighbourhood context token according to Bykau et al. (2015) without any optimisation. We can see that Without optimisation results of our algorithm in 22 is able to achieve better homogeneity and V-measure. Value of completion in both the plot is similar although in case of Bykau et al. (2015) without optimisation it goes to 1 as eps changes from 0.75 to 1. This sudden jump in completion and decrease of value of homogeneity signifies a similar trivial case of all change objects being clustered in one as was the case in our algorithm with $min\_samples = 50$. So without optimisation Bykau et al. (2015) does not perform better than us.
In plots of the results of Bykau et al. (2015) with optimisation, we observe that although our algorithm is still able to achieve better results for homogenity, Bykau

Figure 21: Analysis of extrinsic measures of Completeness, homogenity, V-measure with intrinsic measure of gap entropy for $min\_samples = 50$. X-axis represents different values of eps and y-axis represents the value of the evaluation metrics. Diagram shows negative correlation in shape of gap entropy with extrinsic evaluation measures for all values of $context\_length$. Spike in value of Completeness to 1 is correlated with similar spike in gap entropy. Diagram shows the observation that optimising for values of Completeness leads to drastic decrease in value of Homogeneity.

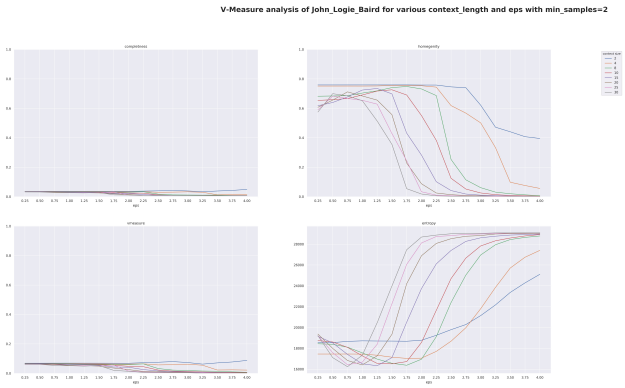


Figure 22: Analysis of our model on Extrinsic evaluation measure of Completeness, homogenity, V-measure with intrinsic measure of gap entropy for $min\_samples = 2$. X-axis represents different values of eps and y-axis represents the value of the evaluation metrics. Diagram shows negative correlation in shape of gap entropy with extrinsic evaluation measures for all values of $context\_length$. All values of $context\_length$ has similar shape and similar value of Homogeneity at maximum point in the graph with respect to eps. Values of Completeness remains considerable with slight increase when homogenity goes down.

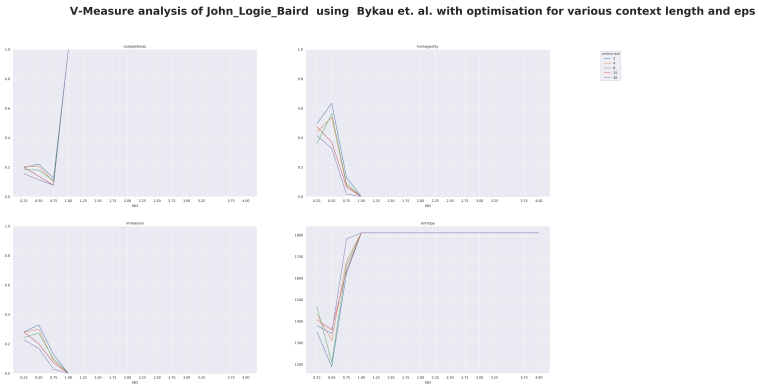


Figure 23: Analysis of Bykau et al. (2015) with optimisation on Extrinsic evaluation measure of Completeness, homogenity and V-measure with intrinsic measure of gap entropy for $min\_samples = 2$. Diagram shows negative correlation in shape of gap entropy with extrinsic evaluation measures for all values of $context\_length$. Values of $context\_length = [2, 4, 8]$ has higher maximum point in the graph with respect to eps compared to $context\_length = [15, 30]$.

et al. (2015) algorithm outperforms our algorithm in Completeness and hence is able to outperform our algorithm in terms of V-measure. One possible explanation of higher completion is that the process of optimisation of cluster by Bykau et al. (2015) is able to separate all the clusters which are not in "nationality" in a separate group. One another observation is that the $context\_length$ of 2,4 and 8 perform considerably better than the $context\_length$ of 15 and 30 inBykau et al. (2015), whereas in our work all $context\_length$ perform similarly. The result is inline with the claim by Bykau et al. (2015) that as context increases more tokens will come, hence introducing noise in Jaccard Similarity. To some extent, we can see this effect in homogeneity of clusters, given by our model, that lower length of context is generally high but is not always true, for example, maximum homogeneity of plot in figure 22 for $context\_length$ 15 is higher than 10. Hence we can conclude that bag of word averaging of embeddings from Mikolov et al. capture better semantics in bigger $context\_length$ than one-hot encoded vectors from Bykau et al. (2015). Finally we can suggest that although Bykau et al. (2015) with optimisation performs better for V-measure but our model is able to better results for all $context\_length$, hence rendering $context\_length$ as less important parameter than Bykau et al. (2015).

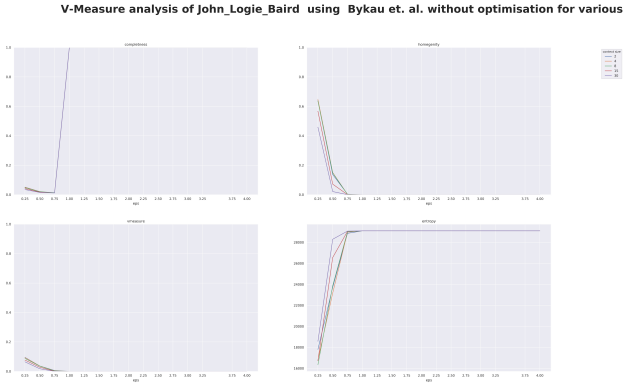


Figure 24: Analysis of Bykau et al. (2015) without optimisation on Extrinsic evaluation measure of Completeness, homogenity and V-measure with intrinsic measure of gap entropy for $min\_samples = 2$. Diagram shows negative correlation in shape of gap entropy with extrinsic evaluation measures for all values of $context\_length$. For all values of $context\_length$ diagram shows the steep increase in gap entropy and steep decrease in Homogeneity. Completeness values are also low and steeply increase to 1 as eps goes from 0.75 to 1.

5.4 Correlation between extrinsic and intrinsic evaluation

In plots of Figure 22, 23 and 24, we can see that the value of the homogeneity of cluster increases with decrease in entropy of gap. This confirms our initial hypothesis that good value of intrinsic measure will help us find better clusters as described in Section (4). We quantify the correlation between gap entropy and three of our extrinsic measures, i.e. Vmeasure, Completeness and Homogeneity. The correlation Table of Figure 25 shows correlations for our clustering algorithm for $min\_samples = 2$. We can see that there is a strong negative correlation between gap entropy and homogeneity. Negative correlation between gap entropy and homogeneity shows that homogeneous clusters have less entropy in gap. In our cluster of $min\_samples = 2$, all the three evaluation metrics homogeneity, completeness and V-measure are strongly correlated as their is very little variation in value of Completeness. Strong negative correlation of three measures with gap entropy is the reason why we were able to get high Fowlkes-Mallows index E.B. Fowlkes (1983) for values of $eps = 1$ pre selected using gap entropy.
Correlation plot in Figures 23 and 24 corresponds to Bykau et al. (2015) with optimisation and without optimisation respectively. Both of them have similar correlation behaviour. Homogeneity and V-measure are negatively correlated with gap entropy but completeness is positively correlated with gap entropy. Homogeneity and V-measure are also negatively correlated with Completeness. Hence for these two measures, Completeness is not a good evaluation metric as its value has high

gap entropy and low V-measure and Homogeneity. Thus Homogeneity is a good measure of extrinsic cluster evaluation index for all the three cases. Homogeneity of our algorithm is more than both of the implementations of Bykau et al. (2015), with optimisation and without optimisation.

6 Discussion

We successfully leverage tracking of each token by WikiWho API to return change objects in the revision history of an article in O(n) time complexity. As WikiWho tokens are unique integers its efficient in terms of memory and comparisons.
We create the clusters of edit tokens in history of Wikipedia edits which are in same neighbourhood context using embedding word vectors from Mikolov et al.. We observe that intrinsic measure of entropy to give us a good measure of quality of cluster. This observation is further supported when we compare agreement between groups identified by our algorithm using $eps = 1$ using token entropy) with the groups identified by Bykau et al. (2015) using Fowlkes–Mallows index. Fowlkes–Mallows index on clusters of 17 articles of varied sizes, as shown in Table 10 with parameter $eps = 1, min\_samples = 5$ have high value for both context sizes of 4 and 10. For both values of $context\_length$ we see that there is an outlier in Fowlkes–Mallows index, on further investigation we see that this is a small article with relatively high amount of Change Objects. As this article should have lots of change object whose context should be similar or with a considerable amount of overlap owing to small size of article and high number of change objects. Thus One possible hypothesis for this behaviour is either ours or Bykau et al. (2015)'s method is not able to separate the change object which are lot closer in vector space than normal articles. Further analysis of article size and number of Change Objects is required to confirm this hypothesis.
Our cluster has higher maximum Homogeneity for all the values of $Context\_length$ compared to Bykau et al. (2015). But in terms of Completeness Bykau et al. (2015) with optimisation is able to capture more complete clusters, which suggests that optimisation is able to remove noise around **ChangeVector**s Hence our model is not able to perform significantly better than Bykau et al. (2015) in terms of V-Measure. Higher values of Completeness in Bykau et al. (2015) suggest a strong dependence on optimisation performed by them when we observe Completeness and V-Measure of Bykau et al. (2015) without optimisation in plot of figure 24 we see a lower value than what we get in our results in plot 22. Although our model gives lower Completeness evaluation score, but it gives slightly better score on Homogeneity with lesser number of parameters, suggesting that our model performs little better by overfitting less for size and kind of article.
$context\_length$ is an important parameter for Bykau et al. (2015), but our model performs similarly for all values of $context\_length$. Plot in 23 shows Bykau et al. (2015) starts introducing noise as value of $context\_length$ increases more than 8, while plot

	change_object_completness	change_object_homegenity	change_object_vmeasure	gap_entropy
change_object_completness	1.000000	0.931869	0.994673	-0.873487
change_object_homegenity	0.931869	1.000000	0.944193	-0.979728
change_object_vmeasure	0.994673	0.944193	1.000000	-0.899761
gap_entropy	-0.873487	-0.979728	-0.899761	1.000000

Figure 25: Correlation of extrinsic and intrinsic evaluation of our clustering algorithm. Matrix shows correlation between intrinsic measure of token entropy and extrinsic measure of Completeness, Homogeneity and V-measure. All the three measure of extrinsic evaluation shows high negative correlation with intrinsic measure of gap entropy.

	change_object_completness	change_object_homegenity	change_object_vmeasure	gap_entropy
change_object_completness	1.000000	-0.798606	-0.857511	0.851718
change_object_homegenity	-0.798606	1.000000	0.988415	-0.984534
change_object_vmeasure	-0.857511	0.988415	1.000000	-0.976839
gap_entropy	0.851718	-0.984534	-0.976839	1.000000

Figure 26: Correlation of extrinsic and intrinsic evaluation of Bykau et al. (2015) with optimisation. Matrix shows correlation between intrinsic measure of token entropy and extrinsic measure of Completeness, Homogeneity and V-measure. Homogeneity and V-measure shows high negative correlation with intrinsic measure of gap entropy. Completeness had positive correlation with gap entropy.

	change_object_completness	change_object_homegenity	change_object_vmeasure	token_entropy
change_object_completness	1.000000	-0.609701	-0.702473	0.673925
change_object_homegenity	-0.609701	1.000000	0.982152	-0.981906
change_object_vmeasure	-0.702473	0.982152	1.000000	-0.988851
token_entropy	0.673925	-0.981906	-0.988851	1.000000

Figure 27: Correlation of extrinsic and intrinsic evaluation of Bykau et al. (2015) without optimisation. Matrix shows correlation between intrinsic measure of token entropy and extrinsic measure of Completeness, Homogeneity and V-measure. Homogeneity and V-measure shows high negative correlation with intrinsic measure of gap entropy. Completeness had positive correlation with gap entropy.

from our model in 22 and 20 shows that all the $context\_length$ has similar maximum value. Thus suggesting that word vectors from Mikolov et al. is able to capture more semantics of words in contexts then finding similarity between exact word. Although more work is required to study the effect of $context\_length$ and extrinsic evaluation metric. A bigger corpus of annotated dataset of ground truth comprising of change object identified across multiple articles of varying sizes and number of change object can be used to investigate the effect of $context\_length$.
Gap of Change Object is an important part of our model using which we use to define gap entropy but apart from that we have only used gap in creating annotation for our golden dataset. In our model, we are able to extend Bykau et al. (2015) by just using cluster in context of gap. Higher values of our model on homogeneity by using the gap and its negative correlation with gap entropy suggests that gap is able to capture semantics of fine grained change object. More work is required to analyse the effect of gap in clustering fine grained change object and introduce more parameters in creating **ChangeVector**s from respective Change Objects.

ChangeVectors consisting of averaged word embeddings from Mikolov et al. perform considerable well for different values of $context\_length$. While each word embedding represents semantics of that word, in order to get average of these embeddings we loose the relative semantics created by order of these words. Performance of these embeddings can be further evaluated by considering different word representation especially the ones which considers context of token in consideration Devlin et al. (2018).

7 Appendix

7.1 Definitions

1. id_i: A unique id representing ith revision of a Wikipedia article.
2. $editor_i$: Editor of the revision corresponding to the id_i.
3. $timestamp_i$: Time at which $editor_i$ created the revision id_i.
4. $str\_token\_list_i$: Ordered sequence of string tokens representing content of article in revision id_i.
5. $wiki\_who\_tokens_i$: Ordered sequence of integral TokTrack tokens from WikiWho API corresponding to each token in $str\_token\_list_i$.

   $$wiki\_who\_tokens_i[j] \in wiki\_who\_tokens_i \implies wiki\_who\_tokens_i[j] \in wiki\_who\_tokens_i. \tag{22}$$

6. $inserted\_mask_i$: Ordered Sequence where each value is true if a token was inserted in that position.

   $$a = [] \tag{23}$$

7. $deleted\_mask_i$: Ordered Sequence of same size as $tok\_track\_id_i$, where each value is true if that token will be removed in next edit.
8. $Revisions$: A time ordered sequence of all revisions of a given article.

   $$Revisions = \begin{bmatrix} r_1 & r_2 & r_3 & \cdots & r_n \end{bmatrix} \tag{24}$$

   $$r_i = \begin{bmatrix} id_i & editor_i & timestamp_i & str\_token\_list_i & tok\_track\_id_i & inserted\_mask_i & deleted\_mask_i \end{bmatrix} \tag{25}$$

9. Change object consists of two ordered list of tokens which were added and removed from same contiguous positions in a revision.
10. $left\_context\_start$ WikiWho token from where left context of Change Object starts.
11. $right\_context\_start$ WikiWho token from where left context of Change Object starts.
12. $context\_length$ Length of token considered in left and right context to encode **Change_Vector**.
13. **Change_Vector** Vector representation of context of Change Object in R^{600} created by averaging of word embeddings.

References

B. T. Adler, J. Benterou, K. Chatterjee, L. D. Alfaro, I. Pye, and V. Raman. Assigning Trust to Wikipedia Content. 2007.

L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179–190, March 1983. ISSN 0162-8828. doi: 10.1109/TPAMI.1983.4767370.

Y. Bengio, R. Ducharme, and P. Vincent. A neural probabilistic language model. volume 3, pages 932–938, 01 2000. doi: 10.1162/153244303322533223.

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, Mar. 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=944919.944966`.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching Word Vectors with Subword Information. 2016. ISSN 10450823. doi: 1511.09249v1.

R. N. Bracewell. *The Fourier transform and its applications / Ronald N. Bracewell*. McGraw-Hill New York, 2d ed. edition, 1978. ISBN 007007013.

U. Brandes and J. Lerner. Visual analysis of controversy in user-generated encyclopedias. 7(2008):34–48, 2009. doi: 10.1057/palgrave.ivs.9500171.

P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, Dec. 1992. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=176313.176316`.

S. Bykau, F. Korn, D. Srivastava, and Y. Velegrakis. Fine-grained controversy detection in Wikipedia. In *Proceedings - International Conference on Data Engineering*, 2015. ISBN 9781479979639. doi: 10.1109/ICDE.2015.7113426.

S.-c. Chin and W. N. Street. Detecting Wikipedia Vandalism with Active Learning and Statistical Language Models Categories and Subject Descriptors.

M. Collins. Language Modeling.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

L. M. E.B. Fowlkes. A Method for Comparing Two Hierarchical Clusterings Author (s): E . B . Fowlkes and C . L . Mallows Source : Journal of the American Statistical Association , Vol . 78 , No . 383 (Sep ., 1983), pp . 553- Published by : American Statistical Association. *Journal of the American Statistical Association , Vol . 78 , No . 383 (Sep ., 1983), pp . 553*, 78(383):553–569, 1983.

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996. URL http://dl.acm.org/citation.cfm?id=3001460.3001507.

F. Flöck, K. Erdogan, and M. Acosta. TokTrack: A Complete Token Provenance and Change Tracking Dataset for the English Wikipedia.

A. Forte and A. Bruckman. Why do people write for wikipedia? incentives to contribute to open-content publishing. 01 2005.

J. Giles. Internet encyclopaedias go head to head. *Nature*, 438:900, dec 2005. URL https://doi.org/10.1038/438900a http://10.0.4.14/438900a.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed <today>].

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of Tricks for Efficient Text Classification. 2016. ISSN 10450823. doi: 1511.09249v1.

D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0130950696.

A. Kittur, B. Suh, B. A. Pendleton, E. H. Chi, L. Angeles, L. Angeles, and P. Alto. He Says , She Says : Conflict and Coordination in Wikipedia. 2007.

Y. Ko. A study of term weighting schemes using class information for text classification. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 1029–1030, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1472-5. doi: 10.1145/2348283.2348453. URL http://doi.acm.org/10.1145/2348283.2348453.

R. Kuhn. Speech recognition and the frequency of recently used words: A modified markov model for natural language. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 1*, COLING '88, pages 348–350, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics. ISBN 963 8431 56 3. doi: 10.3115/991635.991706. URL https://doi.org/10.3115/991635.991706.

S. Lahiri. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 96–105, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E14-3011.

B. Lantz. *Machine learning with R*. 01 2013. ISBN 1782162151.

R. Lau, R. Rosenfeld, and S. Roukos. Adaptive language modeling using the maximum entropy principle. In *Proceedings of the Workshop on Human Language Technology*, HLT '93, pages 108–113, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics. ISBN 1-55860-324-7. doi: 10.3115/1075671.1075695. URL https://doi.org/10.3115/1075671.1075695.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, Dec. 2004. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1005332.1005345.

C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

W. McKinney. Data structures for statistical computing in python. In S. van der Walt and J. Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. Technical report. URL http://ronan.collobert.com/senna/.

T. E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015. ISBN 151730007X, 9781517300074.

J. Pasternack and D. Roth. The wikipedia corpus. 2008.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

F. Pérez and B. E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.53. URL https://ipython.org.

R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

A. Rosenberg and J. Hirschberg. V-Measure : A conditional entropy-based external cluster evaluation measure. (June):410–420, 2007.

R. Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech Language*, 10:187–228, 1996.

R. Rosenfeld. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, Aug 2000. ISSN 0018-9219. doi: 10.1109/5.880083.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov. 1975. ISSN 0001-0782. doi: 10.1145/361219.361220. URL http://doi.acm.org/10.1145/361219.361220.

C. Shanon. The Bell System Technical Journal. *The Bell Sysytem technichal Journal*, XXVII(3), 1948. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6773024.

J. Sivic and A. Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):591–606, April 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.111.

F. B. Viégas, M. Wattenberg, and K. Dave. Studying Cooperation and Conflict between Authors with history flow Visualizations. 6(1):575–582, 2004.

B.-q. Vuong and E. P. Lim. On Ranking Controversies in Wikipedia : Models and Evaluation. 2008.

M. Wikipedia. Measuring Wikipedia. pages 1–12, 2005.

Wikipedia contributors. Wikipedia:lamest edit wars — Wikipedia, the free encyclopedia, 2019a. URL https://en.wikipedia.org/w/index.php?title=Wikipedia:Lamest$_e dit_w ar soldid =$ 884176630. $[Online; accessed 21 - February - 2019]$.

Wikipedia contributors. Wikipedia:article size — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Wikipedia:Article$_s izeoldid =$ 883738489, 2019b. $[Online; accessed 21 - February - 2019]$.

D. M. Wilkinson and B. A. Huberman. Assessing the value of cooperation in wikipedia. *First Monday*, 12(4), 2007. ISSN 13960466. doi: 10.5210/fm.v12i4.1763. URL https://firstmonday.org/ojs/index.php/fm/article/view/1763.

T. Yasseri and R. Sumi. Dynamics of Conflicts in Wikipedia. 7(6):1–12, 2012. doi: 10.1371/journal.pone.0038869.

Y. Zhang, R. Jin, and Z.-H. Zhou. Understanding bag-of-words model: A statistical framework. *International Journal of Machine Learning and Cybernetics*, 1:43–52, 12 2010. doi: 10.1007/s13042-010-0001-0.