INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Introduction to NLP

Prof Manish Shrivastava

Academic Year 2020 - 2021

Unsupervised Chunking for Indian Languages

Author
Chayan Kochar
Tanishq Goel

Abstract

Chunking, or shallow-parsing, is a task that requires the identification of syntactic units which belong together, for example verbs and verbal auxiliaries are one chunk. However, it does not specify their internal structure, nor their role in the main sentence. The aim of this project is to create a phrase based chunking algorithm for Indian Languages.

We have implemented chunking of Indian Languages using unsupervised approach. Although supervised learning algorithms have resulted in the state of the art and high accuracy systems on varieties of tasks in the NLP domain, the performance in source-poor language is still unreasonable. A fundamental obstacle of statistical shallow parsing for the quantities of world's language is the shortage of annotated training data. Furthermore, the work of well-understand hand annotation has proved to be expansive and time consuming. Hindi Language still has relatively decent amount of training data but that can't be said about languages like *pali* or *prakrit*

We have implemented chunking through K-Means Clustering which is a very popular unsupervised ML Algorithm which make inferences from data set using only input vectors without referring to known or labelled outcomes. Main objective of K-means is simple: group similar data points together and discover underlying patterns. Algorithm is discussed in detail further.

Chapter 1

Introduction

1.1 Shallow Parsing

Shallow parsing, also known as light parsing or chunking, is a technique for analyzing the structure of a sentence in-order to identify these phrases or chunks. We start by first breaking the sentence down into its smallest constituents (which are tokens such as words) and then grouping them together into higher-level phrases.

1.2 Unsupervised Approach

Unsupervised Learning is a machine learning technique in which the users do not need to supervise the model. Instead, it allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with the unlabelled data. It allow users to perform more complex processing tasks compared to supervised learning. Although, unsupervised learning can be more unpredictable compared with other natural learning methods. Unsupervised learning algorithms include clustering, anomaly detection, neural networks, etc.

1.3 K-Means Clustering

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from data sets using only input vectors without referring to known, or labelled, outcomes.

The main objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a data set. A cluster refers to a collection of data points aggregated together because of certain similarities. We defined a target number k, which refers to the number of centroids we needed in the data set. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The 'means' in the K-means refers to averaging of the data; that is,

finding the centroid.

How the K-means algorithm works

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids. We chose clusters on the basis of verbs or nouns present in the sentence. It halts creating and optimizing clusters when either:

1. The centroids have stabilized — there is no change in their values because the clustering has been successful.
2. The defined number of iterations has been achieved.

1.4 Hierarchical Clustering

Hierarchical clustering involves creating clusters that have a predetermined ordering from top to bottom. For example, all files and folders on the hard disk are organized in a hierarchy. There are two types of hierarchical clustering, Divisive and Agglomerative.

We implemented it using Agglomerative Algorithm. In agglomerative or bottom-up clustering method we assign each observation to its own cluster. Then, compute the similarity (e.g., distance) between each of the clusters and join the two most similar clusters. Finally, repeat steps 2 and 3 until there is only a single cluster left. Before any clustering is performed, it is required to determine the proximity matrix containing the distance between each point using a distance function. Then, the matrix is updated to display the distance between each cluster.

1.5 Merging

On implementing both the algorithms individually and comparing results from both of them, we got different insights about our model. So we merged both the models taking specific features from both the algo to get better accurate results.

Chapter 2

Execution

2.1 Dataset

For unsupervised chunking, the dataset provided will just be a cleaned corpus with POS tags. Data set can be accessed in the GitHub repository.

2.2 Text-Preprocessing

Basic cleaning and text preprocessing is done for the given corpora.

2.3 Implementation

```
while len(sent_copy1)!=0:
    iter += 1
    initial_length = len(sent_copy1)
    sent_copy = list(sent_copy1)
    for word_ind in sent_copy:
        # print(word_ind,sent[word_ind])
        for cluster in cluster_list[sent_num]:

            # print(cluster)
            if word_ind in cluster:
                sent_copy1.remove(word_ind)

            elif (word_ind == (cluster[0]-1)):
                # print("AAAA")
                temp_pos = [sent[k][1] for k in cluster]
                # print(temp_pos)
                for zz in temp_pos:

                    if re.search(".*N_N.*", zz):
                        if re.search("DM.*|JJ.*|QT.*|RP_INTF", sent[word_ind][1]):
                            cluster.insert(0,word_ind)

                            sent_copy1.remove(word_ind)
```

```
                            break

                    elif re.search("V_.*", zz):
                        if re.search(".*NEG.*", sent[word_ind][1]):
                            cluster.insert(0, word_ind)

                            sent_copy1.remove(word_ind)
                            break
            elif (word_ind == (cluster[-1]+1)):
                temp_pos = [sent[k][1] for k in cluster]
                for zz in temp_pos:

                    if re.search(".*N_N.*", zz):
                        if re.search("PSP|RP_[^INTF]", sent[word_ind][1]):
                            cluster.append(word_ind)
                            sent_copy1.remove(word_ind)
                            break

                    elif re.search("V_.*", zz):
                        if re.search("PSP|.*AUX.*|.*NEG.*", sent[word_ind][1]):
                            cluster.append(word_ind)
                            sent_copy1.remove(word_ind)
                            break

    if len(sent_copy) == initial_length:
        # print(sent_copy1)
        for remain_word_ind in sent_copy1:
            for cluster in cluster_list[sent_num]:

                if remain_word_ind == cluster[-1]+1:
                    cluster.append(remain_word_ind)
                    sent_copy1.remove(remain_word_ind)
                    break
```

2.4 Observations

Initially after implementing the basic algorithm of Kmeans and Hierarchical on some random data, we were confused about on what parameters of text to apply Clustering algorithm to. We observed that word had no role as uch, the only thing for us to play with was RELATIVE INDEX of the words and their POS_TAGS. Thus we basically implemented clustering on the relative positions of the words, using the bsaic structure of Hindi language based on POS tags. Following are some of our observations and outputs according to what we implemented:

1. Sometimes the tag 'NST' is the head of chunk or sometimes inside it. Our model was unable to decode this properly, and considered NST just like other Nouns. For eg: "school ke saamne...." Here 'samne' has 'NST' tag, and is part of the chunk headed by "school", but our model considered it separate.

2. We observed Cases like 'NNP NNP' could not be chunked into one. For example: "Nawab Pataudi" is a one chunk but our model identifies this name as seperate two chunks. So if we have a sentence like "Nawab Pataudi ne ....." - so our model correctly identifies to put the postposition with NNP in Noun Phrase, but it cant chunk between two nouns. Eg: Nawab - B Pataudi - B ne - I .... When we calculated total number of such chunks, we found occurence is quite low so it doesn't affect the accuracy much. We did tried to incorporate NER model of hindi in the same but as accuracy of Hindi NER Model is quite low, this incorporation wasn't feasible.

3. Total possible types of noun chunks and verbs chunks were found along with their number of occurrences. Files could be accessed in the observations directory in the repository.

4. Many tokens, if not were formed part of cluster, were given 'I' tags. Due to this some instances of text like "RP* JJ -¿ JJP " were missed.

5. Many occurences of Noun Phrase chunks occur so rarely that we can avoid that case and altogether ignore that case. Same can be said for the Verb Phrase chunks.

6. Accuracy of Kmeans and Hierarchical Model were 74% and 71%. Our Final Merged Model's accuracy come out to be around 82% which is quite high as it was implemented by unsupervised approach.

2.5 Conclusion

Accuracy of individual model were found and compared. K-means performed better in terms of chunking when compared to hierarchical. However, merged model showed significant higher accuracies when compared to the two individual models. Accuracy of 82% is highly satisfactory.

2.6 Future Work

Further our model could be improved by taking rare occurences of NP and VP into account. Also incorporation of NER model (with better accuracy) will give better results.