Exploiting weak-supervision for detecting and classifying Mandarin Non-Sentential Utterances in Conversations

Chen Xin-Yi

Exploiting weak-supervision for detecting and classifying Mandarin Non-Sentential Utterances in Conversations.

Chen Xin-Yi

Master in Experimental Linguistics
Supervised by : Laurent Prévot

Contents

Abstract

In conversations, there are pervasive short utterances in incomplete syntactic form but that present no difficulty for understanding, these "non-sentential utterances" (NSU) are our target of research in this study. They are short yet efficient in the conversation flow. The interpretation of NSU is relevant for linguistic research as well as for potential industrial applications, like dialogue system. The main tasks they are concerned with are their detection and their classification.

We hope to build a model to classify NSUs automatically. In machine learning, one of the difficulties for building a model is a lack of training data. In this study, we adopt a weak supervision tool, "Snorkel" to perform the automatic classification for the NSUs.

To understand NSU classification, we discuss related literature of dialogue act because it is providing important information about the utterance, and the way their classification is approached in computational approaches is very similar to NSU classification. Also, we look at disfluency because they are short and can be confused with NSU.

Equipped with descriptive statistics of our data and qualitative analysis of NSU categories in our corpus, we attempted both NSU detection and classification within Snorkel framework. We use a set of features based on our corpus in the writing of rules to attribute labels to the data. We can adjust our model based on the performance parameters and error analysis. We summarised the results of our experiments and point directions for future work.

Keywords: *Conversation, Non-Sentential Utterance, Fragment, Weak-supervision, Snorkel*

Abbreviations

AdvP Adverbial phrase

CAP Capability, modal case

CE Clarification Ellipsis

CLF Classifier

COP Copula

DIR Directional

EMP Emphatic

EXCLAM Exclamative

FP Final Particle

LF Labeling function

LOC Locative case

MAN Manner

MOD Modal

NEG Negation, negative

NP Noun phrase

NSU Non Sentential Utterance

NUM Numeral

OBJ Object, objective case

PL Plural

POS Part-of-speech

POSS Possessive marker

PP Prepositional Phrase

PREP Preposition, prepositional case

PTCL Particle

SG Singular

SU Sentential Utterance

SUP Superlative

TEMP Temporal case

TRANSL Translative case (becoming)

Acknowledgments

The master years have been challenging and fun. The sunlight in Aix-en-Provence is very generous, and the croissant and baguette are delicious. The courses of this master are rich, providing comprehensive knowledge for aspects of the science of language, including cognitive science and pathology. I have experienced inspiring moments in classes and with precious people.

This dissertation is the output of the master 2 project I accomplished under the supervision of Dr. Laurent Prévot. I am very grateful for his accepting me to supervise for my Master 2 project when I have to start from almost zero in the second year. He is capable and productive in directorship, patient, and encouraging when I feel discouraged and panicked. He helped students use tools to write the dissertation more efficiently, and besides the essay, he offers priceless help in my pursuit of further projects. He conveys a positive and efficient attitude. For these and many details unsaid, I wish him and his family all the best. Special thanks to Pierre Magistry, who has provided the technical support of the POS-tagging of the corpus used in this study, not only once. I want to thank the responsible person of the Experimental Linguistics Cristel Portes. She has been very professional and caring for our session and offered valuable support along these two years. The crew of all the courses provided by the Science of Language of Aix-Marseille University and Speech and Language Laboratory(Laboratoire Parole et Langage) have shown expertise and showed us their work to enlighten us.

A thankful note to all my classmates/friends, they come from different countries with various personality, I learned a lot from them. During confinement, when I didn't have much physical contact with people or outside world, they offered care and love. Also big thanks to the CROUS Aix Marseille Avignon, a lot of logistic support could have gone wrong during the confinement, but their work makes the international students like me stay safe and assured in France.

Last but least, great thanks to my family, who offered countless support of all kinds, they show optimism in hard times and always encourage me to pursue my dream in faraway places, staying there for me.

Chapter 1

Introduction

Object In dialogue, besides well-formed complete sentences, a sizeable amount of utterances are fragments that could be understood without a problem in their context. Traditional grammar attends mainly to written texts and canonical sentence analysis. The oral language has been often regarded as bad, spontaneous, and wrong, in summary not an appropriate research object, as (Blanche-Benveniste, 1997) regrets it. But as interest in oral communication gets more attention, terms like "fragments", "Nonsententials" in (Barton, 1991) or "Non Sentential Utterances" (hereafter NSU) in (Fernández et al., 2007) have also attracted more investigation.

The expressions in 1 below may sound familiar.

(1) What now?
Not you.
What's for supper? - Ground Beef Tacos.

Justification Even though they are generally short, such utterances constitute an active part of the conversation. They contribute to the efficiency of the conversation flow. Simple feedback can encourage the main speaker to continue the speech. The interpretation of NSU is essential for linguistic theories that attempt to get serious about language as it is produced in its most natural and pervasive setting, and also for applications, like dialogue system. It can be done in different ways, as discussed in (Ginzburg, 2012, p:229), there are approaches solely based heavily on a grammatical mechanism, and other approaches based on pragmatics, involving a strategy of constructionism and dialogue-oriented. The analysis result can be implemented in human-machine dialogue systems in client service, teaching, and other possible uses.

The percentage of NSU in conversation corpus is non negligible, 11.15 % in (Fernández and Ginzburg, 2002), 9% in (Fernández et al., 2007), 10.2 % in (Schlangen and Lascarides, 2003). What's more, the understanding of NSUs and the classification of them from their context is not always easy. Even a simple "what" can express various emotions and can

have different functions in a context. Apart from the most common function as plain question, it can also express Happiness, Surprise, Sadness, Anger, Disgust and Fear, etc. Such as in the example 2 below :

(2) -The winner is ... YOU!
-What?! (Happiness)
-She already left.
-What?!(Surprise)
-What's that strange sound?
-What?! (Fear)

We think the study of NSUs is useful because of the high frequency mentioned above. .

Second, the definition of NSUs can have an impact on the classification, the inclusion and exclusion of categories can be flexible according to the theories and purpose of classification, the classification criteria could be syntactic leading, semantic leading or a mix of standards. The treatment of some fragments like *'Greetings'* and *Filler'* can make a difference in the counts. We will see the detailed discussion in 2.

Methodology Fragments are not uniform, so to classify them correctly is not always instinctive. In this dissertation, we choose a semi-supervised approach. Why semi-supervised? First, because supervised learning needs enough labeled data so aside from expertise, the annotation often requires a good deal of time and money. Second, unsupervised learning does not need explicit labels, but the results may be hard to interpret and can't satisfy specific research goals like classification. Thus, we decide to combine linguistic rules along with unsupervised learning to build a classification model.

Several factors are influencing the classification of NSUs. First, there are different aspects in linguistic analysis, such as syntactic, semantics, and pragmatics in analyzing these small pieces, according to the research question and their focus, the theoretical framework chosen can differ. We will see later that some studies are more focused on semantics, and some are more prone to syntactic structure, for example. Second, according to the purpose of classification, the granularity of classes can differ. It will enrich the theoretical analysis but also can be helpful in the application.

In chapter 2, we will review some different classifications and generally, they have two layers, the first one is some big categories, and then there is a various number of subcategories. Third, there could also be language-specific behavior; for example, following the work of (Fernández et al., 2007), the work of (Wong and Ginzburg, 2013) in classifying NSUs in Chinese adds seven subcategories because of the particular behavior of modal verbs in Chinese. We will also look at some examples of Spanish in (Garcia-Marchena, 2015) and of French in (Pénault, 2018).

In this study, we adopt a semi-supervised learning method in the classification of NSU, and we use a weak supervision tool Snorkel[1] (Ratner et al., 2017b) which combines algorithmic labeling system and domain knowledge. We hope to build a model to classify NSUs automatically with satisfying results. We feed the model with a small amount of labeled data; it will generate an extensive training set that could be used in machine learning.

Research Questions What is proper classification of NSUs for Chinese mandarin conversation corpus? What is the distribution of different classes? Using this weak supervision technique, what are the useful clues to detect and classify the NSUs as accurately as possible?

Contributions Our scientific contribution is to test and develop schemes for describing NSU in Mandarin on a conversation corpus. NSU related work is in the minority compared with that of dialogue act, both in detection and classification, and only a few works have been done regarding Chinese. The classification we choose is (Wong, 2018), which is based on the work of (Fernández et al., 2007) in adding some classes considering particular behaviors in Chinese Mandarin. Our corpus is a telephone conversation corpus, CALLHOME Mandarin Chinese Transcripts[2]. Another difference is that Taiwan Mandarin can present some differences with Chinese Mandarin usage in Mainland China so that we might find some gaps in our corpus. Considering the large population and the relatively few studies, we think it is valuable to explore the NSU features in Mandarin Corpus.

As for methodological contribution to the NSU classification task, we use Snorkel, a weak-supervision system that can generate a large amount of training data from a small set of labeled data. The system provides a solution to a bottleneck in machine learning, to train models, training data is needed. But the way to obtain them is costly in time and money, and domain experts are not always available, the crowd-sourcing labels are not always trustworthy. Using Snorkel combines the linguistic expertise and machine-learning mechanism to produce labeled data effectively. We hope that this approach can be useful in exploiting knowledge of domain expertise, and especially linguistic knowledge, in a more efficient way than large annotation campaigns. Through automatic annotation design for high-level tasks that cannot be addressed without annotated data.

Plan Chapter 2 presents the related work of utterance classification, including dialogue acts and NSUs (Non-Sentential Utterances)With an overview of these classifications, we decide on the applicability of these classifications in the mandarin corpus. Chapter 3 introduces the data and methodology. The weak supervision machine learning tool Snorkel generates training sets automatically with appropriate labeling functions. Chapter 4 provides a qualitative description of the NSU instances of each category in our corpus. We present several general ways of labeling functions and what we use in our analysis. Chapter

[1] https://www.snorkel.org/

[2] (https://catalog.ldc.upenn.edu/LDC96T16)

5 is about the detection task of NSU, on identifying the discrepancy between NSU and sentential utterance and disfluency, we use these indexes to build a model to sort the NSUs from the corpus. Chapter 6 talks about the classification experiment in our work, what the features are, and how they are used in classifying tasks. The results and evaluation of the experiments will be presented at the end of chapter 5 and 6. Section 7 concludes the dissertation.

Chapter 2

Related Work

2.1 Introduction

To understand and explore *Non Sentential Utterances* (NSU), we need to talk about some related nottions : Dialogue act and the closely speech act, also some adjacent concepts like feedback and back-channels.

Since (Austin, 1962), "to say something is to do something.", understanding an utterance is composed of two related aspects: understanding its content, which consists of the literal meaning of the utterance; the other aspect is understanding its function: the message conveyed by the utterance, "I'm hungry." can be a description of a state of fact or a demand for getting some food or even some kind of complaint. According to (Austin, 1962), the illocutionary function co-occur with the locutionary act(the act of making a meaningful utterance) and presents the possibility of interpretation beyond the utterance's literal meaning.

The participants in a conversation constantly update the shared information and their beliefs. In (Stalnaker, 2002), the mutual beliefs is called *Common Ground*. In (Bunt, 1995), the functional unit used in changing the context is called "dialogue act", these units are not equal to utterances, because utterances are generally multifunctional. The notion of dialogue act (DA) remind people with "speech act" of Austin, but dialogue act is more closely related to computational linguistics. (Bunt, 2017) explains the differences between speech act and dialogue act, speech act theory of (Austin, 1962) and (Searle, 1969) are within the philosophy of language and it is an approach to meaning. In contrast, for Bunt, *"the theory of dialogue acts is an empirically-based approach to the modeling of linguistic, nonverbal, and multimodal communicative behavior."*

As stated in (Bunt, 2017), feedback is the processing result of the interlocutor's previous utterance in a dialogue. This is a complete result, including perceptual, interpretation, evaluation, and possibly other cognitive aspects. Feedback is realized through various specific linguistic, paralinguistic, and nonlinguistic means. It is also explained in (Bunt,

2017) that feedback could be positive or negative, the negative feedback express difficulties in processing the previous utterance. Positive feedback can be expressed by common words like "ok" or "yes", paralinguistically by "Mmm, "ah," and by some gestures like nodding, laughing, etc. In (Muller and Prevot, 2009), it's pointed out that the positive feedback (or acknowledgment) is said to be often mixed with backchannels. In (Harvey, 2011) backchannels are words or phrases that provide feedback to the dominant speaker by indicating that the non-dominant speaker is still engaged in the conversation (though not actively participating at the moment).

NSU is an utterance that is not realized by a full syntactic sentence. It produces an effect, just like sentential utterances. Even though All utterances can receive a DA label, but only those fragments with incomplete syntactic structure and full semantic value can be labeled as NSU. As shown in 3. All NSUs are speech/dialogue act realizations, but not vice versa.

(3) A : Who killed Bill?
B1 : John killed Bill. (Speech Act = Statement, Dialogue Act = Answer, NSU : No)
B2 : John. (Speech Act = Statement, Dialogue Act = Answer, NSU : Yes)
B3 : John? (Speech Act = Question, Dialogue Act = Confimation-Request, NSU : Yes)

A dialogue act annotation system is multidimensional; this is not the case for the NSU. Characterizing an utterance for DA says something both about the semantic content type and communicative function. But for the NSU, it could also be influenced by other factors, such as syntactic form.

As disfluency cases are confusing when we look at NSU cases, both are not canonical full sentence, the two tend to have a short duration, fewer words, and not always predicates. Also, the disfluency can receive DA tags but are separated from NSU. We tend to do a discussion in subsection2.2.5 to elucidate the difference.

In the following sections, we will discuss dialogue acts in 2.2, the definition and the taxonomy and some models of dialogue act, and also a look at the feedback and back-channel research, and discussion about disfluency. In 2.3, we will compare four different classifications of Non Sentential Utterances for English, French and Spanish.

2.2 Dialogue acts

2.2.1 Definition

We are interested in the dialogue act in this study because of their relationship with NSU. It is summarised in (Levinson, 1983) that "speech acts should be defined according to the modification effects on the context." So it is with dialogue act. DA is about the meaning

at the illocutionary level defined in (Austin, 1962), which is the intent or effect produced along with the things being said.

(Bunt, 1995) first defined DA as "functional units used by the speaker to change the context", this is very similar to the idea of speech act. (Bunt, 2005) pointed out an essential difference between the speech act and dialogue acts, which is that the utterances are considered to be multifunctional, that an utterance could be used to perform multiple dialogue acts, but in speech act theory embraces a unique speech act for an utterance. Besides, in (Bunt, 2017) it is further stated that the speech act theory is concentrated on verbal behavior while dialogue act englobes nonverbal and multimodal behaviors too.

Besides its importance in linguistic analysis, this term has become visible in the research on computer dialogue system design or dialogue annotation. In (Bunt, 2005) it's explained that DA has an extensive range of application scenarios. The DA labels demonstrate the hidden information of the utterance for higher-level processing. It can be used in the interpretation and generation and prediction of utterances and their functions in dialogue systems, as stated in (Stolcke et al., 2000). Therefore DA-tagging is a major applicative task for NLP annd Human-Machine Interaction. An example of Dialogue Act adapted from (Stolcke et al., 2000) is presented in Table 2.1.

Speaker	Dialogue Act	Utterance
A	YES-NO-QUESTION	Do you like listening to music?
A	ABANDONED	Or yo-
B	YES ANSWER	Yeah,
B	STATEMENT	I listen and I compose music every week.
A	DECLARATIVE-QUESTION	You're a , so you are an expert.
B	YES ANSWER	Yeah,
B	STATEMENT	I hope to be an artist one day.
		[laughter].
A	APPRECIATION	Oh, good for you.

Table 2.1: Example of DA adapted from (Stolcke et al., 2000)

2.2.2 Taxonomy of Dialogue acts

A DA taxonomy has to make a compromise between two requirements according to (Fišel, 2007). First, the DA tags need to be as clear as possible, so people don't have to worry about which tag(s) to attribute; second, a taxonomy should be as general as possible for different corpus and purposes.

In some studies an existing taxonomy is used, but in others the author creates his(her) own taxonomy. A widely accepted mark-up is Dialogue Act Markup in Several Layers [DAMSL] (Core and Allen, 1997), from which a taxonomy can be derived. As said in (Fišel, 2007), *"The dialogues are annotated on four different levels, which are the communicative status, the information level, forward-looking function and backward-looking function."*

(Fišel, 2007) mentioned another widely used taxonomy is based on DAMSL, designed for the Switchboard corpus and is called SWBD-DAMSL. The two have 80% overlaps of tags This tag set is multidimensional, the annotators used around 220 unique results. Then, as said in (Stolcke et al., 2000), to achieve a better agreement rate among annotators, they downsized it to recognize 42 mutually exclusive utterance types, some of them are illustrated in Table 2.2.

The tag set represents a mixed standard, incorporating both form-based labels and discourse-theoretic related labels, and it was designed so that the annotators do not need audio files, so just the text transcription when deciding about the tags as explained in (Stolcke et al., 2000).

The tags distribution is skewed, the major dialogue act types are: Statements and Opinions, Backchannels, Turn Exits and Abandoned Utterances, Answers, and Agreements. Below is a part of the table of tags distribution for the 42 classes, and the rest is less than 5%.

Tag	Example	%
Statement	Me, I'm in the legal department.	36%
Backchannel /Acknowledge	Uh-huh	19%
Opinion	I think it's great	13%
Abandoned / Uninterpretable	So, -/	6%
Agreement / Accept	That's exactly it.	5%

Table 2.2: Part of tag distribution in a 42-classes dialogue act set

2.2.3 Dialogue-Act Classification

Two primary sources of information are used to classify dialog acts in (Stolcke et al., 2000): Hidden Markov Models (HMM) and N-gram discourse models (sequences of previous dialogue act). The goal of the (Stolcke et al., 2000) is "to perform DA classification and other tasks using a probabilistic formulation"; it involves the probability laws and inferential statistics in the combination of multiple knowledge sources and for automatic derivation of model parameters. With all the given information about a conversation, they aim to find the DA sequence with the highest posterior probability.

They also used acoustic cues but the design of the tag set allows an accuracy without audio, the acoustic evidence include "various aspects of pitch, duration, energy, etc." and spectral features used in ASR.

Lexical and prosodic cues are both useful for the dialogue act classification, as we don't use audio resources in our study, I will put aside the prosodic part here. It is observed that some words are symbolic of some DAs. For example, in (Stolcke et al., 2000), "92.4% of the uh-huh's occur in Backchannels, and 88.4% of the trigrams '<start> do you' occur in Yes-No-Questions." For some shared patterns, the differentiation is by pronunciation.

As mentioned in (Fišel, 2007), for the feature data types, the three most common types are binary, numerical, and nominal. Binary features are often coded as 0/1 as a scalar

value. “Nominal features are most commonly encoded as vectors, composed of binary indicators, each corresponding to a possible feature value.”

An important kind of DA is feedback / back-channel. They are very frequent, as can be seen in table 2.2, and are of high relevancy to our study since most of the feedback and backchannels take the form of extremely short utterances that lack a sentential flavor. . In terms of dialogue act dimension, which corresponds to types of information, and defined in (Bunt, 2006) as "an aspect of participating in dialogue", feedback is also a significant type. These dialogue acts offer or evoke information about the processing of previous utterances as explained in (Bunt et al., 2010).

Due to the close relationship between the Dialogue Act and NSU, we have discussed DA in this subsection, as well as the definition of DA, its taxonomy and its classification. An essential type of DA is feedback/backchannel, specifically the backchannel, which is short and corresponds to our interest in NSU. We will look at them in detail in the next section.

2.2.4 Feedback and Back-channel research

Back-channel as considered as a speech act-like unit in (Stolcke et al., 2000), this small unit has important discourse-structuring roles. Thus, detecting a backchannel may be useful in predicting utterance boundaries and surrounding lexical material." These are usually referred to in the conversation analysis literature as "continuers" and have been studied extensively. "The term backchannel first appeared in (Yngve, 1970) when talks about the conversation parties engaged through speaking and listening, *"This is because of the existence of what I call the backchannel, over which the person who has the turn receives short messages such as 'yes' and 'uh-huh' without relinquishing the turn."* (Tao and Thompson, 1991) also agrees that is the producer of backchannel does not claim the floor (if the interlocutor doesn't finish his talking), typical English non-lexical verbal expressions such as *aha, uh-huh, mhm, yeah.* "The Mandarin counterparts are forms such as *ao, a, ei ; and dui/shi('right, yes').* In English, typical expression used in backchannel feedback include yeah, uh-huh, hm, right and okay.(add a reference) In Chinese, frequent backchannel feedbacks such as "嗯",(Um) "哦"(oh).

While in (Xudong, 2008), the non-lexicality in form and passive recipiency in sequential function is emphasized in backchannel, it is further clarified that the lexical items should be excluded. In Chinese Mandarin, the “dui”(‘right’) and “shi ah”(‘yes + vocalic particle’) are excluded. And only those showing recipiency are backchannel. (Sacks et al., 1974) such as answer to question, it's not an backchannel. Last, the evaluative or expressive words such as “wow” in English and “ai ya” “wa” in Mandarin will not be taken into count. Thus typical backchannels in mandarin are : hm , mhm, ah, oh, oh ah. each of them can be repeated or combined.

It is also stated in (Stolcke et al., 2000) that short DAs like ANSWERS and BACKCHANNELS "tend to be syntactically and lexically highly constrained". So the detection of these tags can be rather efficient using limited cues. In (Jurafsky et al., 1998), they investigated the realization of five dialog acts which are included in the backchannel act : continuers,

assessments, incipient speakership(indicating the producer of the utterance want to take the floor), agreements and a subtype of answer category, yes-answers. Because they have great overlap in the lexical indices they use, like "yeah, okay, uh-huh, or mm-hmm", so it's hard to decide which is the right act.

It is raised a "cue word" hypothesis in (Jurafsky et al., 1998): "while some utterances may be ambiguous, in general the lexical form of a DA places strong constraints on which DA the utterance can realize." It has been proved that the distribution of these words in each category is differentiated. For instance "uh-huh is twice as likely as yeah to be used as a continuer, while yeah is three times as likely as uh-huh to be used to take the floor". This is informative for similar tasks in differentiating similar tasks for tags in our work. To see the distribution of the key words in similar categories.

But it's to be noticed that other factors may also have an effect like individual difference, some may tend to use the same particular words for some act while others may use the set of words in varying their intonation for different functions.

Feedback and backchannel have essential roles in discourse construction. They are often one-word units signaling the processing of information from the main speaker. We have seen frequently used words for feedback in Mandarin in English; their connection with DA as well as restrictions in its definition. Different words have different distributions for various functions. But even with the same terms, if there is a repetition of several words or cases of truncated feedback, this is not a matter of feedback anymore but more a disfluency, as we are going to see it in the next subsection.

2.2.5 Disfluency

We want to discuss the disfluency because they share some characteristics with NSU, and are often short utterances as well. We hope to find some distinguishing qualities between them and NSU to separate them for our task successfully, and the theoretical discussion will help us understand why and how.

In (Blanche-Benveniste, 1997), it is mentioned the difficulty of transcribing oral language, due to its three characteristics: Paradigmatic overcrowding, back and forth on the syntactic axis, and insertion as shown in example 4 translated from French in (Blanche-Benveniste, 2010, p:27-30). And the behavior of the speaker can be an obstacle or "precious clues" in our work. We distinguish three kinds of behavior: unfinished, self-corrections, and conversation effect.

(4) Paradigmatic overcrowding:
which was fantastic in that-
in this-camping
in that
this hotel is that we were totally
Back and forth on the syntactic axis & insertion:
Its purpose is to give, uh, create new systems
new mechanical systems.

It is further explained in (Blanche-Benveniste, 1997) that some unfinished phrases are deliberate, leaving unsaid or completed by another. Some indicate a hesitation, searching of words, or scraps that could be abandoned or finished later in the speaking flow. This kind of behavior counts for a frequent disfluency case in our corpus.

In (Blanche-Benveniste, 1997) about self-corrections, often, the corrections are accompanied by repetitions of some related element, for example, if the speaker first said "some fruits" and then corrected as "some oranges", then our repeated item is the quantifier.

(Shriberg, 1996) talked about several types of disfluency: *filled pause, repetition, substitution, insertion, deletion and speech error* as illustrated in 2.3. This classification is more detailed than that in the (Blanche-Benveniste, 1997), for example, the filled pause that stands on its own from other kinds of insertion.

Later (Tseng, 1999, p:39-41), in a discussion about how to detect repairs, Tseng distinguishes two possibilities:

- one is based on a hypothesis of edit signals, which assumes that a edit signal always marks the location of self-corrections, using the signal as a reference point, before is a stack of incomplete nodes and after is complete constituents. So with the combination of these edit signals and grammar rules this should work. But the edit signals aren't fixed expressions in utterances, we should look at the data to determine the signal.

- Another possibility is to match patterns. For example, in the work of (Heeman and Allen, 1994) , pattern-machine was used in detecting speech repairs, by combining the repair pattern rules with the statistical model filter. It involves three types of speech repairs: fresh starts, modification repairs and abridged repairs, as demonstrated in 5. Fresh starts are when the speaker starts over and abandon the original production. Modification repairs involve a similarity between the reparandum and the changed version. For abridged repairs, there is no reparandum, only a fragment or editing term. The three types of speech repairs were annotated inset but with a similar system but with additional POS information.

(5) Fresh start:
I think we need to - okay it would be better if we have a negotiation .
Modification repair:
I need to book the flight to — the flight with AirFrance.
Abridged repair:
We need to -um book the flight right now or we won't get the discount.

In (Tseng, 1999)'s exploration of modeling the disfluency, the corpus is a conversation corpus in German. There are features found to be useful in the detection of disfluency on the syntactic side: the linguistic length, the syntactic category, the construction types, the location of interruption, the repair onset, and the repair offset. These features could be useful in our examination of disfluency in our corpus.

The majority of disfluencies consist of more than one phrase length. The phrase is defined according to conventional grammar. Prepositional and noun phrases account for 75% of all disfluencies found in the corpus. Further, there is a correlation between syntactic structure and disfluency type. The interruption tends to come after the determiners.

In (Tseng, 2003) 's research about repairs and repetitions in spontaneous Mandarin, the editing part is in the middle between the reparandum and the reparans. Reparandum refers to the "mistaken" part to be replaced, Editing term is an indication of speech repair such as "well" "I mean" or filled pauses. Reparans designates the "corrected" version of the reparandum as in example 6. In contrast with partial repetitions and repairs, for which over half occur with an editing term, complete repetitions are usually without editing term, because no new information is produced, so an editing phase is unnecessary. Thus the editing term can be used in the detection of these two kinds of disfluency. Besides, as there are some NSU categories involving repetition such as repeated acknowledgment, this could be compared with the repetition in disfluency.

(6) Could you buy me some oranges-, no, some limes?

Reparandum: some oranges
Editing term: no
Reparan: some limes

We have seen in this subsection the categories in disfluency and the detection method and modeling of disfluency based on their qualities. We hope this will be of assistance to compare the similarities and, more important the differences with the NSU qualities, which we will see in the next section.

2.3 Non Sentential Utterances

We use the classification based on (Wong, 2018), which inherits the grouping in (Fernández et al., 2007). We will make a comparison of other classifications and discuss briefly their theoretical backgrounds.

In this section, we discuss the definition of Non-Sentential Utterance (NSU) and the different related concepts; we compare in particular existing taxonomies. We start from the taxonomy of (Fernández and Ginzburg, 2002) and we choose four representative classifications, the taxonomies in (Schlangen, 2003), (Fernández et al., 2007) and the research concerning fragments in oral Spanish in (Garcia-Marchena, 2015) and study regarding fragments in oral French in(Pénault, 2018). We take the fragments as the equivalent of NSU in this study.

Our focus is the detection and the classification of NSUs. We will mention the theoretical bases but we will not deliver deep theoretical discussion here, as we are more

interested in the operationalisation of taxonomies for machine learning identification and classification experiments.

There are two kinds of perspectives of fragments. These units smaller than a sentence and also may not be complete in terms of the predicate. The traditional term has been "fragment," one way to look at it is ellipsis analysis, that this kind of structure is an ellipsis from a full sentence, that some part has been omitted. Such an approach presupposes a sentential analysis of a non-sentential utterance.

The second approach considers this kind of units as well-formed, and argues "against the assumption that all lexical categories automatically occur as complements to functional heads." in (Barton, 1991). In the latter perspective, the focus is put on different level of analysis: discursive relations such as in (Barton, 1991), macro-syntactic in (Pénault, 2018), semantic coherence in (Schlangen, 2003).

2.3.1 Classification of Fernández and Ginzburg

An earlier version of (Fernández et al., 2007) taxonomy was introduced in (Fernández and Ginzburg, 2002). The categories of the 2007 version are already all present in the 2002 version. But in the corpus study, only 12 classes were found and discussed. The whole set of classes can be decided through four decision trees (DT) as illustrated in 2.1.

The first DT decides which DT the NSU arrives next, DT-Q if the NSU is a question, DT-A if an answer, DT-O if neither a question nor an answer. In DT-Q (*Questions*), the sub-questions include whether the question implies uncertainty about the antecedent, then the presence of a "wh-phrase", last, is there confirmation of understanding. In DT-A (*Answers*), it questions about what kind of answer it corresponds to, it could be a "wh-question," a polar question, and the answer could be affirmative or negative, there could be repeated part in the response and it could include an alternative. In DT-O (*Others*), there are two sub-categories, one goes into acknowledgment and the other concerning extension moves englobing all kinds of modifiers.

The decisions in the decision trees are designed to be simple enough for the non-expert to provide spontaneous answers. After all layers of questions, each class has its place in the trees and the overall classes are represented in figure 2.1, we can see there is also a class for "other" showing this classification can be non-exhaustive.

The (Fernández and Ginzburg, 2002) is supposed to be the first "comprehensive, theoretically grounded classifications of NSU in large-scale corpus". The classification is based the British National Corpus (BNC), the classification take into consideration both a relatively complex syntax and the context dynamics. In the 2007 version (Fernández et al., 2007), several machine-learning experiments were carried out to get an optimal classification result. The features selected for Machine Learning in this article is limited in a few "meaningful" ones instead of many arbitrary ones. The features selected for NSU classification included, one is about the utterance itself, the second is about its antecedent, the

Disfluency class	Example
filled pause	I. ~~uh~~ recommend it
repetition	~~I~~ . I recommend it
substitution	~~I~~ . we recommend it
insertion	I ~~recommend~~ . really recommend it
deletion	~~I am kind of~~ . I recommend it
speech error	~~me(err)~~ I recommend it

Table 2.3: Table adapted from (Shriberg, 1996)

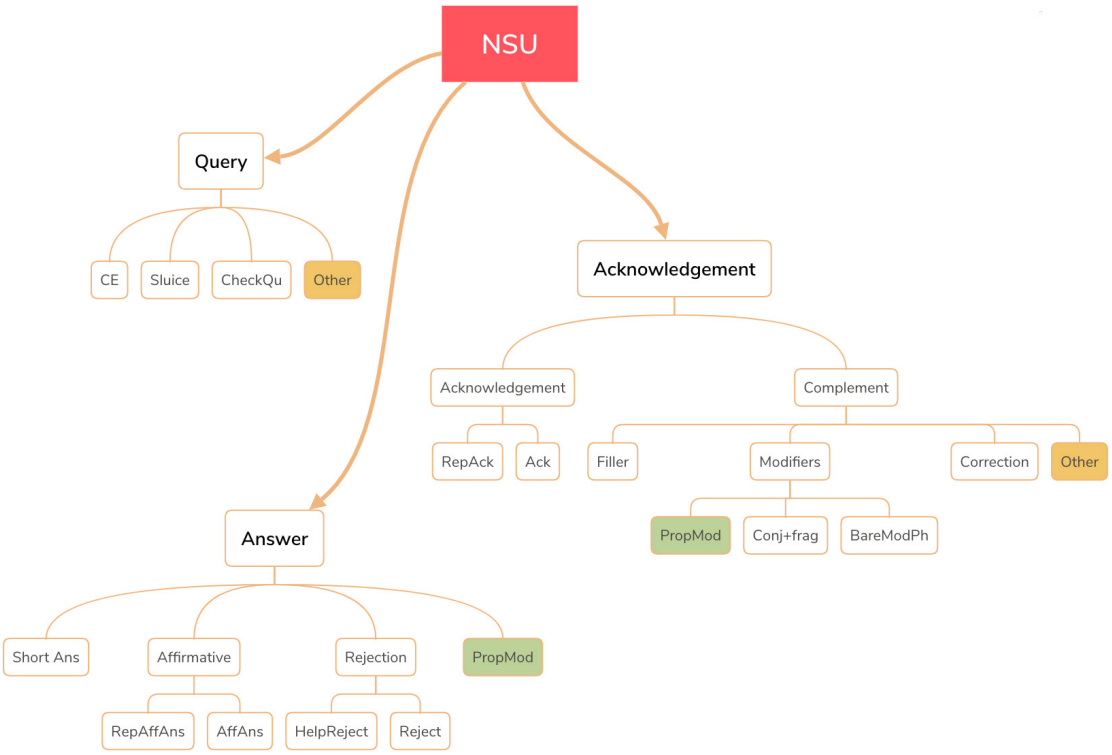


Figure 2.1: Adapted NSU classes affiliations based on the decision trees in (Fernández and Ginzburg, 2002)

third one is about their relationship. Three sets of features in total: *NSU features, Antecedent features and Similarity features*. The NSU features include four aspects, whether it is proposition or question, presence of wh-word, yes/no word, and different lexical items. The antecedent features are similar to those of NSU features, but it also looks at whether it's a finished utterance. The similarity features is a comparison of the utterance and its antecedent, mainly about the repeated words and POS tags and their proportion. We will look at them more specifically in chapter 5. Another machine-learning experimentation work for classification of NSU is based on the work of (Fernández et al., 2007) with more advanced features in (Dragone, 2015). We'll look in detail in chapter 5 when presenting our selection of features.

2.3.2 Classification of Pénault(2018)

Pénault(2018)		Fernández et al. (2007)	
First level classes	Second level classes	Second level classes	First level classes
Repetition	Agreement	Repeated acknowledgement	Acknowledgement
	Comment	Plain acknowledgement	
		Propositional modifier	
		Factual modifier	
Addition	Modifier	Filler	
		Helpful rejection	Answer
	Question	Sluice	Query
Replacement	Correction	Helpful rejection	Answer
	Precision	Helpful rejection	
	Answer	Short answer	
	Elaboration requisition	Clarification ellipsis	Query
		Check question	

Table 2.4: Comparison of classification between (Pénault, 2018) and (Fernández et al., 2007)

(Pénault, 2018) distinguished four types of NSUs in her corpus. They are *fragments with linguistic sources*, *fragments with extralinguistic sources*, *simple prosentences* and *prosentences with contrastive topics*. Besides, there are two levels of classification, the first level concerning the semantic input from the target statement to its source. The second level includes eight kinds of semantic and/or interactional functions. Going through the first level of classification, we get eight types of fragments. In table 2.4, we summerized correspondences between the classification of (Pénault, 2018) and (Fernández et al., 2007) that will study in detail with examples in the next pages.

Repetition : agreement, comment

The definition of Repetition type in (Pénault, 2018, p:137) is "Semantic content redundancy consists in the *verbatim* or *propositional anaphora* of content already present in the context". According to this definition, these two classes corresponds separately to

REPEATED ACKNOWLEDGMENT and PLAIN ACKNOWLEDGMENT in F&G(2007)'s classification, whose precedent is an narrative sentence and not a question. The example of agreement and comment in (Pénault, 2018, p:137) :

(7) A : Tu fais euh un petite boucle vers le bas B : vers le bas (MTX)
A: You do uh a little loop down B: Down (MTX)

A : Le chat dort. B : oui (EF)
A: The cat is sleeping. B: yes (EF)

Addition : modifier, question

As explained in (Abeillé et al., 2007), for a functional definition of sentence, an expression is analysed as sentence if its head is a predicate saturated for its subject. The predicate could be itself or its content based on the extension of the definition. (Pénault, 2018, p:137) states the addition/ completion could be syntactic and a supplement when the source's argument structure is already saturated. It also includes self-completion or self-correction and competitions by others. These two classes can correspond to several types of NSUs in F&G(2007).

The modifier in (Pénault, 2018) can be FILLER in (Fernández et al., 2007) when the previous speaker did not finish his/her sentence, as the example below in (Pénault, 2018, p:140).

(8) A : tu prends à heu B : à gauche (EF)
A: you take to uh B: left (EF)

On discussing the semantic function, three functions are mentioned, apart from the modifiers and question, there is also a function *comment*. The modifiers can be used in both conditions whether the linguistic aspect is saturated or not. Observed from examples, this class could be Corrections in its classification, but as stated in the article, the plurality of categories is accepted. So it could be HELPFUL REJECTION in F&G (2007) classification as the example showed in (Pénault, 2018, p:138).

(9) A : j'ai les balises à droite j'ai d'autres balises à droite B : en dessous (MTX)
A: I have the beacons on the right I have other beacons on the right B: below (MTX) (EF)

The questions with interrogative pronouns are equivalent to *sluice* in F&G(2007) classification, like the example 10 in (Pénault, 2018, p:140). The comment class matches the modifier class in F&G (2007), both the PROPOSITIONAL MODIFIER and FACTUAL MODIFIER which in typical forms of Prepositional phrase and/or Adverbial phrase, see the example 11 in (Pénault, 2018, p:140).

(10) A : il est arrivé hier B : avec qui (EF))
A: he arrived yesterday B: with whom (EF)

(11) A : Jeanne a eu son bac B : fantastique (EF))
A: Jeanne got her baccalaureate B: Fantastic (EF)

A : L'eau bout à 78 degrés C : n'importe quoi (EF)
A: Water boils at 78 degrees C: nonsense (EF)

Replacement: correction, precision, answer, elaboration requisition

Like addition in (Pénault, 2018), the replacement can also be self-replacement or hetero-replacement. There is a limiting condition that the element replaced and replacing one should be of the same syntactic category.

The response class can be Short Answer class in F&G(2007); the precision can be Helpful Rejection, but it restricts the type of precision that it should be a hyponym of a part of the previous utterance. The correction can also be Helpful Rejection in F&G(2007), but a "no" is not necessary; The elaboration requisition can be Clarification Ellipsis or Check Question in F&G(2007). We present the series examples by order of mention below, taken from (Pénault, 2018, p:140).

(12) A : Quelle heure est-il ? B : neuf heures [EF]
A: A: What time is it? B: Nine o'clock [EF]
(Response - Short Answer)

A : c'est un canidé B : un fennec [EF]
A: It's a canine. B: A fennec. [EF]
(precision - Helpful Rejection)

A : c'est un canidé B : un félin [EF]
A: It's a canine. B: A feline. [EF]
(correction - Helpful Rejection)

A : C'est un mollusque B : un quoi [EF]
A: It's a mollusk B: A what [EF]
(elaboration - Clarification Ellipsis or Check Question)

The repetition class indicated an anaphora, so the fragments which have non-linguistics source are not included. The addition type is compatible with both linguistic and non-linguistic fragments. The replacement type also involves only linguistic pieces, which also implies an anaphora.

The second level of classification in (Pénault, 2018) is related to the semantic function of the utterance compared with the source. In sum, with the proposed eight classes of NSU, there are four absent classes compared with the fifteen types in F&G(2007). They are PLAIN AFFIRMATIVE ANSWER and PLAIN REJECTION, BARE MODIFIER PHRASE and CONJUNCT. It is essential to remind that (Pénault, 2018) purposefully excluded some categories such as PLAIN FEEDBACK such as "hm" and stand-alone interjections and discourse markers from her study since she focused on syntactic structure. When the utterances are only made of one "lexical item" that did not interact syntactically with the context, they tend to be excluded.

Even though there is no discussion about the computational application about this classification, the comprehensive analysis may be useful, like the summary of the sequence types in the corpus and their corresponding sequences (Pénault, 2018, p:142). Below in Table 2.5 is the nominal sequences examples translated from the original table, these may be useful to serve as features using the POS tags for our machine learning models.

Constructions	Sequences	Examples
NOMINALS		
Composed of one or more names (N)	N	"misunderstanding" [MTX]
	NN	"dreadful horror"[DVD]
Composed of noun(s) and adjective(s) (A) and/or a determiner (D) and/or adverb (R)	AAN, AN, NA, ANA DAN, DN,DNN, DNA, DA ,DNRA	"the 'thin' stuff"[MTX]
		"a palm"[MTX]
		"two centimeters"[MTX]
		"a very epic passage"[MTX]
		"the bullshitters besides"[MTX]
Composed of a name and a prepositional group (S)	NSN	"dispute of neighbourhood" [CID]
Containing a pronoun (P)	PN	what radio [DVD]
Including an interjection (I)	IDN, IN, IN, DNI,NI	"her head, uh" [DVD]
		"uh the oasis" [MTX]
		"Uh, wisteria"[CID]
		"the left one ok "[MTX]

Table 2.5: Morpho-syntactic data construction types

2.3.3 Classification of Schlangen(2003)

The study (Schlangen, 2003) is about interpreting the fragments. In (Schlangen, 2003, p:3), it's said that the resolution of the intended meaning of fragments can be modeled as a by-product of establishing the dialogue's coherence, and creating consistency depend on and interacts with both linguistic and extra-linguistic information sources. The linguistic constraints include syntactic, semantic, and pragmatic restrictions that are some conventional limits on the well-formedness, the extra-linguistic knowledge can be world-knowledge or plans. .

Also, it makes some exclusions of categories. First, it's assumed that fragments are intended to be in an incomplete form, those deficient due to production error(disfluencies such as repetitions, self-repairs) are not qualified as NSU. Second, there are two kinds of NSUs not discussed in the article, even though they meet the requirements for "fragments". The first is due to the situation and without antecedent. Such as "A croissant, please".

This has a much higher demand for context knowledge for comprehension. The other kind is mostly the greetings, bearing function of "discourse management" in (Bunt, 1994). Requests are also excluded in the codification of (Schlangen, 2003, p:18) because the focus is on "relations with propositions and questions as arguments."

In the classification of (Schlangen, 2003, p:14), the first layer of classification includes only two dimensions. The primary criterion is the functional difference or puts differently, the instantiated speech act of the fragment. The second criterion is the material needed in the interpretation of the fragment. Thus there are two types of resolution, *resolution-via-identity* fragments and *resolution-via-inference*. The difference lies in whether we can interpret the meaning just from the given previous utterance, or we need some extra information to infer such as general background information.

Compared to our study, a different point in (Schlangen, 2003) is its classification is based solely on the order of utterance, but no precision or emphasis of the producer of the utterance. So one or some sequences of statements could be produced by the same speaker, as indicated in some examples in (Schlangen, 2003), where there is only one speaker. At the same time, in (Pénault, 2018, p:43), it is stated clearly that the classification of NSUs should be dialogical, we too are interested in the dialogical situations.

It is said in (Schlangen, 2003, p22) that the tags are not mutually exclusive, yet in (Fernández et al., 2007), it's the other way around. Considering the purpose of our classification, we are not sure that we need such fine-grained classes for question-related classes. So we will look at some but not all classes in later discussion.

Question-Answer-Pair As the name indicated, this answer is the target of analysis. This could be any sub-categories in the type ANSWERS as well as ACKNOWLEDGEMENT and even MODIFIER phrase in F&G taxonomy.

Elaboration Elaboration is defined as below in Schlangen's and SDRT related work (Asher and Lascarides, 2003; Prevot, 2004). This type describes an utterance in the form of a question, elaborating on some aspects of its indicative antecedent, providing details of events, or providing information about the event's participants.

In this case, the fragment is a question, it corresponds to the SLUICE and CHECK question in the F&G(2007) classification. But sometimes the examples are not a fragment. Instead, it is a complete sentence, the examples in (Schlangen, 2003, p:27) are :

(13) A: I gave her a solemn promise.
B: That you'll never sing in the shower ever again?

A: I made a purchase.
B: The ugly hat you're wearing?

We don't count these sentence examples as NSUs in our corpus. For cases that can be regarded as fragments, and when the fragment aims at a wh-question, we can easily connect it with the SLUICE class in F&G classification.

When both the previous utterance and the fragment are both questions, in this case, the fragments can be identified as CLARIFICATION ELLIPSIS if there is a reprise part of the antecedent. SLUICE and CHECK QUESTION are also equivalents in F&G's glossary. Examples in (Schlangen, 2003, p:29):

(14) A: Did you go to the cinema?
B: With Peter?

A: Did you go to the cinema with Peter?
B: Peter Miller?

Correction Correction is about a partial or complete revision of the facts introduced by the previous utterance, or incomplete or total modification of purpose or concreteness. It's similar to the Correction and Precision in (Pénault, 2018).

Also, it concerns focus change between the precedent phrase and the correction fragment, and the corrected element replace the focus in the previous utterance. At the same time, in a full-sentence form, a non-focused component can be corrected. It corresponds to the HELPFUL REJECTION in F&G (2007).

(15) A: Mike takes [Julie]$_F$ there.
B: No, Jack.
The full-sentence form would be :
A: Mike takes [Julie]$_F$ there.
B: No, [Jack]$_F$ takes Julie there.

The research was fed into a system RUDI (Resolving Underspecification using Discourse Information) built upon the output of ERG, the analysis provides some additional information for the ERG result. The system is base on some decision trees, where data goes through branches of questions. And the temporal indices are used and also syntactic and semantic index. At that time, it deals only with NP- and PP- fragments.

2.3.4 Classification of Garcia-Marchena (2015)

Compared to (Schlangen, 2003), which works on an approach for coherence interpretation for NSU, and mainly about the realized speech act and discourse function realized. (Garcia-Marchena, 2015, p:330) comments on its limitation for its lack of conclusion concerning the syntax of the fragments. Other works are limited in one or certain kinds of fragments. It acknowledges the taxonomy of (Fernández and Ginzburg, 2002) and its application for the Spanish corpus. But it also points out the limitations of it; one is the hybrid of the

classification standards, that different classes can be separated from each other based on various criteria. For example, the different modifiers are different in their semantic content, and the conjuncts stand out for their syntactic structure. This mixing nature causes an overlap of classes for NSU in some categories; for example, the answer type can have a modifier function in some cases. Such as "-Do you know whether he will come to the party? -Probably not." It would be SHORT ANSWER in (Fernández et al., 2007), but it's also a FACTUAL MODIFIER.

Once an utterance received its label in the higher place of the tree, it does not need to go down for other judgment. So according to (Garcia-Marchena, 2015, p:332), this decision is motivated by the method used to identify the fragments, not by its inherent properties.

The discussion of fragments in (Garcia-Marchena, 2015) divided the fragments into two types: the root fragment and the subordinate fragment. We only discuss the previous type here because in our opinion the subordinate fragment in (Garcia-Marchena, 2015) are not real fragments. Instead, these subordinate fragments are often sentences with a main clause and subordinate clause or with a verb and a relational pronoun as illustrated in examples 16.

(16) A ver qué tal. (CON 035A)
See how it is.

Digo yo que sí, que encantada.
I say yes, it's nice to meet you.

While the fragments of the F&G's classification is mixed linguistic objects, and the fragments in (Schlangen, 2003) are discursive semantic objects. In (Garcia-Marchena, 2015), the fragment is defined as a syntactic structure with propositional semantic content, the same as sentences. But there is an asymmetry in the fragments' structure, syntactically it is partial while semantically it is complete. It is explained that the fragments are elliptical phrases, and they are autonomous because of their context. In the example 17 below, the "Estupendo"(Great) is an adjective in the syntactic aspect, but its semantic interpretation include the precedent phrase, expressing an integral meaning as *"It's great that you are going home"*. Example 17 of a fragment from (Garcia-Marchena, 2015, p85).

(17) a. A : -Me voy a casa. B : -Estupendo.
a. A : -I'm going home. B : - Great.

In (Garcia-Marchena, 2015), the fragments are regarded as a parallel concept as sentences, while "utterance" is seen as a pragmatic or discursive object. Also, in (Garcia-Marchena, 2015, p:200), fragments are distinguished from nonverbal phrases in comparing the syntactic structure, the predicative averbal sentence is composed of a non-verbal predicate, plus the argument that saturates it. In example 18, (a) is an example of averbal phrase and (b) is a fragment. The fragment has a predicative head selecting an argument

in deep anaphora. Nevertheless, the invisible or deep anaphora cannot be identified in the syntactic structure.

(18) a. Precioso el lanzamiento. (DEP 017A) 'Beautiful the launch.'
b. Precioso lanzamiento. (DEP 017A) 'Beautiful launch.'

With this asymmetry, it is summarised the difference among the four classes according to the parameters in table 2.6.

Fragment	Provided content	Retrieved content	Anaphora
Illocutionary	Argument	Participant and speech act	Deep
Modal	Modal predicate	Arguments	Deep or surface
Modifier	Argument	Predicate and arguments	Surface
Argumental	Coindexed argument	Predicate and arguments	Surface

Table 2.6: Content type and parameters for fragment type in (Garcia-Marchena, 2015)

There are three classes in the first layer. They are illocutionary fragments, modal fragments and modifiers & argumental fragments. The illocutionary fragments can be further divided into presentative fragments, expressive, promissive, directive, and performative. All the Spanish examples in this section is selected from the article (Garcia-Marchena, 2015) and translated.

The classification of (Garcia-Marchena, 2015, 335) demands an autonomous propositional content of the fragments, so it excluded some classes in the F&G's taxonomy: the FILLER (because it's dependent on the previous utterance), REPEATED AFFIRMATIVE ANSWER, HELPFUL REJECTION, and the REPEATED ACKNOWLEDGMENT. Besides, it rules out the categories which did not have content and only realized a speech act: the CHECK QUESTION and the ACKNOWLEDGEMENT. Moreover, the SLUICE and the joint fragments (the NSUs introduced by conjunctions) are also absent because only their syntactic structure defines them.

The syntactically saturated classes like affirmative and negative responses are also not recognized as fragments. The argumental fragments encompass the reply and repeated kind of fragments in the F&G's taxonomy because they are only defined by the speech act they express. The PROPOSITIONAL MODIFIER class in F&G(2002) express a modal predicate, so they are admitted as counterparts of Modal fragments (epistemological and evaluative) in (Garcia-Marchena, 2015).

With the criteria of fragments in (Garcia-Marchena, 2015) and considering all these exclusions, we look at some categories in (Garcia-Marchena, 2015) that is comparable to those in (Fernández et al., 2007).

Illocutory fragments

In (Garcia-Marchena, 2015, p:402), it's explained that if we have A and B as interlocutors, the Presentative is A INFORMs B about his or her status; the Expressive is A express a

WISH to B; the Promissive is A OFFERs something to B; the Directive is A REQUIREs B of something; the Performative is A GRANT something to B.

Modal Fragments

This kind of fragments expresses modal predicates in two different ways: epistemological and appreciative or evaluative. Epistemological predicates select propositional contents, while evaluative predicates can select entities or propositional contents.

Epistemological In example 19, epistemological fragments are illustrated (from (Garcia-Marchena, 2015, p:445)).

(19) **Presentatives**
a. Claro / Efectivamente / Por supuesto. (CON 023A/CON 004B/CON 029D) 'Sure / Indeed / Of course.'
b. ¿Seguro ? (PUB 034A) 'Are you sure ?'
d. Exacto / Correcto / ¡Pues lógico ! / Seguro al 100%. (ADM 004A/INS 004A/CON 019A) 'Exactly / Right / Well, that's logical ! / 100% safe.')

The epistemological fragments can enter into the MODIFIER class in the F&G's classification, they express different kinds of certainty of the speaker.

Evaluative fragments The appreciative or evaluative fragments are also similar to the MODIFIER class in F&G's classification, like the BARE MODIFIER PHRASE in 20 (a,c) and PROPOSITIONAL AND FACTUAL MODIFIERS(propositional adverbs and factual adjectives) in (Garcia-Marchena, 2015, p:443). We see immediately that most of these are monological, it corresponds to deep anaphora.

(20) **Evaluatives**
a. ¡Vaya pierna ! (LUD 030A) 'What a leg !'
c. ¡Vaya un lío ! (ENT 001E) 'What a mess !'
a.A:-Soy uno de los niños que más duerme la noche y el día.(...)B:-¡Qué suerte ! (ENT 007D)
A : -'I'm one of the kids who sleeps more at night and during the day. (...)B:-'What luck !'

Modifier fragments

In (Garcia-Marchena, 2015, :473), the modifier fragments can be declarative or interrogative, so they are not counterparts of the MODIFIER category in (Fernández et al., 2007) and they can carry out various illocutionary effects (clarifications, recapitulations, ordinations, etc.) What's more, they can appear in multiple configurations: alone, with a fragment, or in a sentence, as an incident, modifying a word or a phrase, as illustrated in 21.

(21) **Modifiers**
¿Contigo ? / ¿Dónde ? / Trabajando ? (CON 023A/CON 022B/CON 021A) ' With you ? / Where ? / Working ? '
Con toda brevedad. (POL 009A) 'Briefly' ('With all brevity')
Por eso. (ADM 004A) 'For this.'
Por decirlo de alguna manera. (DEB 010A) 'In a manner of speaking.'

Argumental fragments

This kind of fragments is characterized by addition plus replacement of the content. CORRECTION and CHECK QUESTION in (Fernández et al., 2007) seem to be the similar to the examples 22 below.

(22) a. A : -Vamos a casa de 'María. B : -De Laura. A : -'A : -Let's go to 'Maria's house. B : -Laura's. '
b. A : -Vamos al centro. B : -¿A casa de Juan? A : -'-Let's go downtown. B : -To Juan's house? '

This work has an implementation in the design of a conversational agent. Although it didn't specify the details, in our view, the in-depth syntactic analysis of each type can be essential as in (Garcia-Marchena, 2015, p:410), including the type of fragment, their head along with the speech act, and the semantic structure. They can be adapted to the analysis of monological as well as dialogical utterances.

2.4 Conclusion

For the Dialogue Act, its taxonomy is instructive for that of NSU. The classification models are also beneficial for NSU related work. The backchannel and feedback are also very frequent in NSU, but not disfluency even though they share some similarity in form. Through the comparison, we can understand the unique quality of NSU more clearly and use them in our experiments.

Based on the review of different classifications of NSU, with the deviation in definition and focus of research, we find some similar categories but also some criteria of exclusion. The non-mutual exclusivity of labels is not overall respected. Our choice of taxonomy for this study is based on Fernandez's work, but there are also other options if we want to deal with tasks fitted with a similar theoretical framework in one of those others. We will look at our corpus now and the method for thoughts of application.

Chapter 3

Data and Methodology

This chapter is to discuss the data and methodology used in this study. We will start by introducing the weak-supervision tool Snorkel, and then we will present the corpus we used.

Both for linguistic analysis and application, labelling/enriching data with linguistics is crucial; that's also the reason for DA's popularity because they provide additional information about the raw material.

But labeling data is very costly and cannot be performed on large data sets, especially with the growing volume of the corpus from different origins. There are ways to get cheaper labeled data from crowdsourcing, but low-quality labels can harm the model.

A tool that weakly supervised tools like Snorkel allows for quickly labeling extensive data with minimal but expert manual involvement. In manual annotation, it is rather common to find repetitive cases for one class, and the regularities may be in different aspects like semantic, syntactic, or its relations with other utterances. We can communicate these rules to the tool we use to do the repetitive work for us.

The use of Snorkel is to write some labeling functions (LF) to produce some useful training data with labels. A labeling function is a rule that attributes a label for some subset of the training data set. Using Snorkel, it will train a model that combines all the rules defined written to estimate their accuracy, along with the overlaps and conflicts among different labeling functions.

The framework for experimenting is Jupyter Notebook; it is a web application where we can create and share documents containing codes, equations, visualizations, and text. It allows real-time data analysis. Jupyter notebook supports many programming languages such as Julia, Python, and R. We use Python 3.7 in this study.

3.1 Automatic Annotation

Machine learning (ML) has been a hot topic in recent years, mostly for its calculation potential for big data and for those ML systems which demand "extensive feature engineering and model specification, leading to confusion about where to inject relevant domain

knowledge." as stated in (Ratner et al., 2017a). Still, it's not easy to use them. Snorkel makes it easier to combine machine learning and domain knowledge with its workflow and user-guides for experts in the various domains to adapt it to their specific needs.

In principle, Snorkel builds and manages training data programmatically without manual labeling, in our case, we have still used a small labeled data set to see the accuracy and develop *labelling functions*. A labeling function is a rule that attributes a label for some subset of the training data set as illustrated in Figure 3.1.

```
@labeling_function()

def has_speaker(x):
    """If several continuous utterances is produced by the same speaker, then it's not a feedback"""
    if x['Same_speaker'] == True:
        return SU
    return ABSTAIN
```

Figure 3.1: Example of LF in Snorkel, SU refers to sentential utterance

The general annotation task presents difficulty and demands for expertise on a continuum. Our mission is to label our data with NSU tags. We make it a two-step task in this study, for each utterance, the initial judgment is whether it is an NSU or not, then decide to which NSU class it belongs. For the detection of NSU in the corpus, based on the results of descriptive statistics in the next section, we can apply these results, such as n-grams of words and POS tags, timing cues and words count, and the transcripts itself into the LF such as in figure 3.2.

Using Snorkel, it will train a model that combines all the rules defined written to estimate their accuracy, along with the overlaps and conflicts (explained in subsection 3.1.2) among different labeling functions. We now look closely at Snorkel's philosophy and how we make use of it.

3.1.1 Snorkel's philosophy

A large quantity of training data is necessary for machine-learning tasks. But labeled data are not easy to get. Snorkel provides a solution to this bottleneck by using labeling functions to generate a large amount of labeled data. As stated in (Ratner et al., 2017b), based on theories and experiments, Snorkel has proven effective in training high-accuracy machine learning models, even using potentially lower-accuracy inputs.

The workflow of Snorkel is special and distinguishes from traditional machine-learning approaches; it is based on a data programming paradigm. We do not go deep about data programming here. Briefly, it is composed of two phases, and the first is to produce estimated labels using a generative model, the second is using these labels to train the ultimate model, a discriminative model.

```
from snorkel.labeling import labeling_function


@labeling_function()
def duration(x):
    return SU if x['Duration']  > 4  else ABSTAIN


@labeling_function()
def latency(x):
    return SU if x['Latency']  > 4 else ABSTAIN


@labeling_function()
def overlap(x):
    return SU if x['Overlap']  > 2 else ABSTAIN

@labeling_function()
def wordcount(x):
    return SU if x['Word_count']  > 7 else ABSTAIN
```

Figure 3.2: Example of LFs to detect SU

What is the difference between a generative model and a discriminative model? When we face some unseen data X, we want to know if it is NSU or not. For a discriminative model, it models the decision boundary between NSU and the others (including Sentential Utterances (SU) and disfluency), and it decides right away which side it belongs. In a generative model, there is no such boundary, it calculates the joint probability distribution of X and the NSU, and also the joint probability distribution of X and other classes, and see which probability is more significant, then we assign it to that bigger class. In other words, there are two camps, and X enter the two camps separately and see which fits better.

Within this design philosophy, the system design of Snorkel can be divided into three phases: first, preprocessing, it can "automatically represent the input data as a hierarchy of unstructured contexts", as showed in figure 3.3, stated in (Ratner et al., 2017b), it's useful for Relationship extraction. In our case, we have preprocessed the corpus based on our needs, and we will explain in section 3.2.

Second, writing labeling functions. For each input, there are three possible outcomes: TRUE, FALSE, and NONE label encoded numerically as 1, 0, and -1. Three approaches are commonly used in writing LF: heuristic pattern matching, distant supervision, and ensembling of weak classifiers. The heuristic pattern matching in Snorkel can be realized through regular expression or more complex and even with external libraries. This is used in our study, with keywords matching along with regular expressions.

Labeling functions do not need to be entirely accurate or exhaustive and can be correlated. Snorkel will automatically estimate their accuracies and correlations in a provably consistent way, as introduced in (Ratner et al., 2016) and illustrated in figure 3.4. Based on the results given(the meaning of the parameters are explained in section 3.1.2), we can improve and abandon the LFs, and do error analysis to see the reason why a certain LF performs poorly.

After the evaluation and calibration of the LFs, we decide on an optimal set of LFs to produce a set of labels to train a model. In the detection task, we use the classifiers from Keras (Chollet et al., 2015) and Scikit-Learn (Pedregosa et al., 2011). But other libraries such as PyTorch, Ludwig, and XGBoost are also possible. They will give a result of test accuracy.

3.1.2 Multi-class classification mechanism

For the NSU detection task, binary labels with abstain are sufficient. While for the classification task, we have far more than two classes, so it is a multi-class task.

In the detection task, we import our preprocessed data, set the labels as $NSU = 1$, $SU = 0$ and $ABSTAIN = -1$. We have a small set of manually labeled data as reference, divided to serve as a development set, test set, and validation set. We write the relative LFs and generate a label matrix using the LFApplier. Then we can see the summary statistics for the LFs as the example in figure 3.4. Here is an explanation of the parameters cited in Snorkel documentation:

- Polarity: The set of unique labels this LF outputs (excluding Abstain)
- Coverage: The fraction of the dataset the LF labels
- Overlaps: The fraction of the dataset where this LF and at least one other LF label
- Conflicts: The fraction of the dataset where this LF and at least one other LF label and disagree
- Correct: The number of data points this LF labels correctly (if gold labels are provided)
- Incorrect: The number of data points this LF labels incorrectly (if gold labels are provided)
- Empirical Accuracy: The empirical accuracy of this LF (if gold labels are provided)

The most direct number is in the column Incorrect and Correct. We can do error analysis by using a helper method **get_label_buckets(...)** to group data points by their predicted label and their true label (the label was given in the development set). By seeing the results of the mislabelled data, we can improve our present LF. By the index of the LFs (the orders of our LFs in the notebook), we can select the columns based on certain LFs and see which labels are misrecognized as illustrated in figure 3.5.

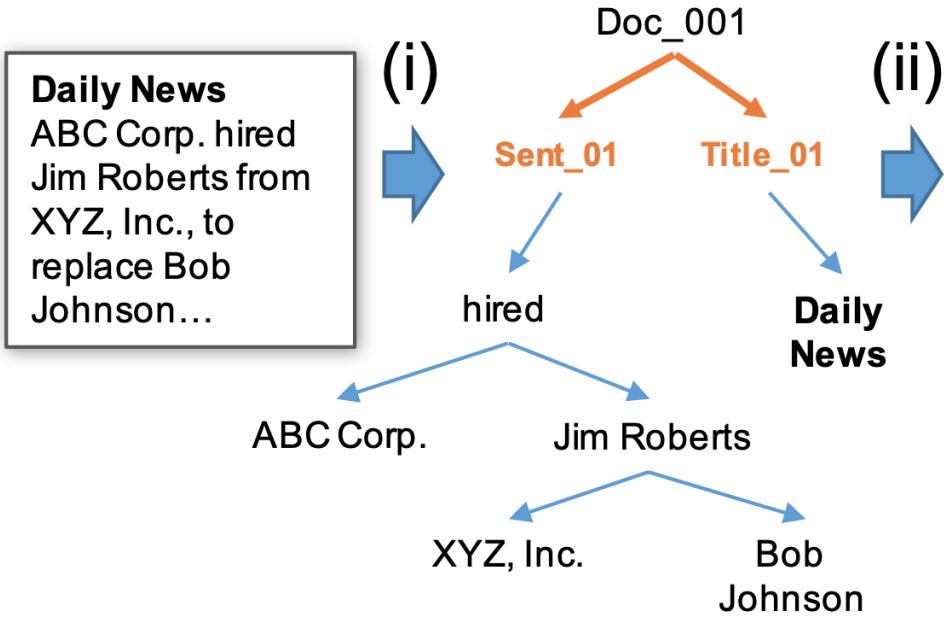


Figure 3.3: Example of preprocessing in Snorkel, image from (Ratner et al., 2017b)

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
pos2	0	[0]	0.127778	0.127778	0.011111	109	6	0.947826
pos_heavy	1	[0, 1]	1.000000	0.788889	0.184444	749	151	0.832222
pos_light	2	[0, 1]	0.737778	0.737778	0.153333	518	146	0.780120
has_speaker	3	[0]	0.197778	0.197778	0.067778	130	48	0.730337

Figure 3.4: Example of matrix with indicators of LFs such as accuracy, conflicts

```
buckets = get_label_buckets(Y_dev, L_dev[:, 1])
df_dev.iloc[buckets[(NSU,SU)]]
```

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Latency	Overlap	Label	Tagged
72	73	882	355.50	359.67	A	呃 , 呃 哼 , 呃 呃 呵 呃 呵 , ((呃)) .	9	4.17	True	NaN	NaN	1	呃_AD ,_PU 呃_NN 哼_VA ,_PU 呃_AD 呃_VV 呵_NN 呃_AD 呵...
242	243	1711	428.75	429.95	A	对 对 对 对 对 , 就 这 意 思 .	8	1.20	False	0.14	0.33	1	对_P 对_P 对_P 对_NN 对_VV ,_PU 就_AD 这_VV 意思_NN ._PU

Figure 3.5: Example of error analysis using the helper method [get_label_buckets(...)]

We can evaluate the balance of the coverage and accuracy of the LF set after seeing the performance on the training and development sets. When we have two good LFs, and one is conflicted with another, we need to make adjustments and keep the better one. If we only look at the numbers, it can be misleading; we need to inspect the labeled data and results of the LFs, some wrongly constructed LF may look good in numbers, but they may hurt the real useful LFs.

The next step is to combine the labeling function outputs with the Label Model in Snorkel, the technical details of the LabelModel is in (Ratner et al., 2016), and in (Ratner et al., 2018). At this step, it is also possible to do error analysis when we get the trained model. Then we remove datapoints receiving no labels from any LFs before training a classifier, otherwise, it will damage the performance.

```
from sklearn.linear_model import LogisticRegression

sklearn_model = LogisticRegression(C=0.001, solver="liblinear")
sklearn_model.fit(X=X_train, y=preds_train_filtered)
```

```
LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
print(f"Test Accuracy: {sklearn_model.score(X=X_test, y=Y_test) * 100:.1f}%")
```

```
Test Accuracy: 67.3%
```

Figure 3.6: Example of result with Scikit-Learn classifier

For the detection task, we have used Keras classifier with Probabilistic labels and Scikit-Learn. The kind of results obtained are illustrated in figure 3.6.

For the classification task, most parts are similar. We import the preprocessed data and assign the labels, except we have more than twenty labels for all the classes. We write LFs and decide on the final LF set along with error analysis and the indicators in the matrix, then use the LabelModel. But in this multi-class setting, we get an F1 Micro average value reflecting the result considering the total true positives, false negatives, and false positives. Besides, we only use a Scikit-Learn classifier.

3.2 Data

Extensive conversational data are limited in numbers. We are interested in the real-time conversational data in talking form transcripted in textual form instead of data generated in textual form in instant-messaging tools.

In (Fang et al., 2019), there is a review of Chinese interactive speech resources. Compared with the written language database, the spoken corpus devoted to linguistics is in the minority. And there is also disequilibrium among languages, even though the number of Chinese speakers is large, there are fourteen Chinese interactive speech corpora established so far, and annotations about these corpora are also a huge amount of work according to the purpose and nature of the corpus.

(Fang et al., 2019) summarised that among the 14 interactive spoken Chinese corpus, due to the different purpose of constructing the corpus, the languages involved in these corpora is not limited to Mandarin Chinese, but also some dialects. For example, the Chinese Language Resource Audio Database (CLRAD) of (Yu-min, 2010). Besides, the content of the spoken corpus includes not only spontaneous conversation but also read speech and narration.

(Fang et al., 2019) also sums up the general features of the currently available Chinese interactive speech corpora: first, the Chinese interactive speech corpora consist of dialogues in different channels, face-to-face conversations, telephone conversations, and Internet chats; second, the language covered including Mandarin as well as dialects; third, the scenario of dialogues include general topics of daily life (in the majority), as well as some specific domain like hotel and flight reservations. Fourth, the form of storage like audios and transcriptions, videos are also possible. Fifth, the applications of these resources are in more and more diverse domains, like speech engineering, dialogue system, automatic translation, etc.

The data we used in this study is from LDC's CALLHOME Mandarin Chinese collection. This is a telephonic conversation corpus, with audio files and transcriptions. The language was in Mandarin even though the participants are from different provinces of China. The recruited participants can make a free call from North America to their relatives or close friends at home in China, and each call lasts for half an hour. This project was done in 1996 when the telephone fee is rather expensive. The corpus includes 120 transcripts in total, and each is a five or ten-minute segment from the telephone speech files. From the description on the website [1], the transcripts are already tokenized automatically using a tool called the Chinese Lexical Analysis System (ICTCLAS). The results were further corrected manually in some aspects, like the four aspect markers in Chinese (-le, -guo, -zhe and -zai) and the Chinese classifiers based on an annotation scheme on the Lancaster University project. Also, a large number of typographical errors were corrected.

[1] https://catalog.ldc.upenn.edu/LDC2008T17

The original data includes the start time and end time of every segment, the speaker (A, B or B1, etc.), the textual content of the utterance. In the text transcription, there are also examples of annotation as enrichment of information as illustrated in 23.

(23) Examples of annotation from http://shachi.org/resources/661
{text}: sound made by the talker. e.g. {laugh} {cough}
{sneeze} {breath}
//text//: aside (talker addressing someone in background) e.g.//quit it, I'm talking to your sister!//
(( )): unintelligible; can't even guess text.

We processed the data and transformed it into Pandas DataFrame (McKinney, 2010) in order to manipulate it into Jupyter Notebooks (Pérez and Granger, 2007). They are transformed as a table, and the information is divided by columns. The original information is separated into four columns: *Start time*, *End time*, *Speaker*, and *Text transcription*. Based on these, we added other columns (illustrated in figure 3.7):

- Conversation code: the original code of the file
- Duration : how long the utterance lasts
- Same Speaker: if the utterance is produced by the speaker of the previous utterance (BOOLEAN)
- Latency : a gap between two turns, the Start time minus the previous End time (we only consider the positive value cases)
- Overlap : the duration when more than one person speaks (we only consider the positive value cases)
- Word count: How many units are there in the utterance (depending on the segmentation method, one unit may not necessary correspond to one Chinese character, and the punctuation can be included as well)
- Tagged: the POS tagged text used in this study is attributed by the tool Zpar[2] (Zhang and Clark, 2011)

3.2.1 Descriptive Statistics of the raw data set

General figures of the dataset

We have 33485 utterances in total in combining 120 files. Combined with the tagged results, we omit the ones untagged, so we deal with 33431 utterances.

[2]Many thanks to Pierre Magistry for performing this tagging.

ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	Start	Text	Word_count	Tagged
1	882	1.73	185.20	NaN	NaN	False	A	183.47	你 很 爽 哈 你 , 哈 ?	6	你_PN 很_AD 爽_AD 哈_VV 你_PN ,_PU 哈_VV ?_PU
2	882	1.02	185.95	NaN	0.27	False	B	184.93	要 不要 跟 妈 讲话 ?	5	要_VV 不_AD 要_VV 跟_P 妈_NN 讲话_VV ?_PU
3	882	0.42	186.45	0.08	0.00	False	A	186.03	啊 ?	1	啊_VA ?_PU
4	882	0.28	187.17	0.44	0.00	False	B	186.89	啊 ?	1	啊_VA ?_PU
5	882	10.78	199.50	1.55	0.00	False	B1	188.72	&乖乖& , 哎 小 &朱& 实在 是 很 有 有 福气 . 他 一直 在 盼 着 说 , ...	42	&乖_NR 乖&_VV ,_PU 哎_AD 小_JJ &朱&_NR 实在_AD 是_VC 很...

Figure 3.7: Head of the preprocessed corpus dataframe

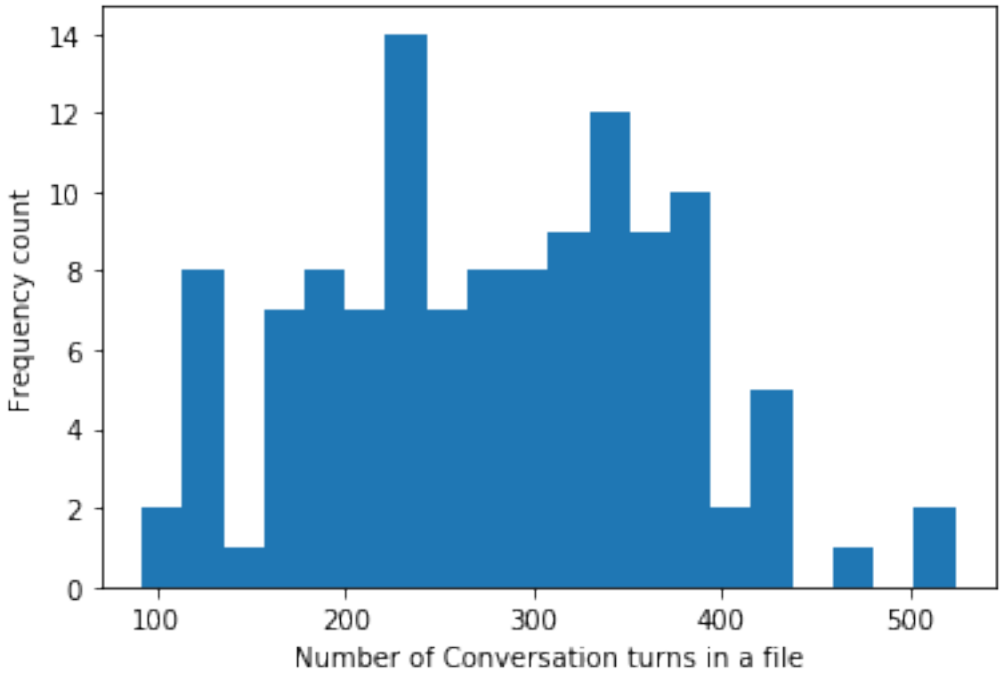


Figure 3.8: Conversation Turns per file

We can see in figure 3.8 the number of utterances among the 120 files, most of them are between 200-400 turns.

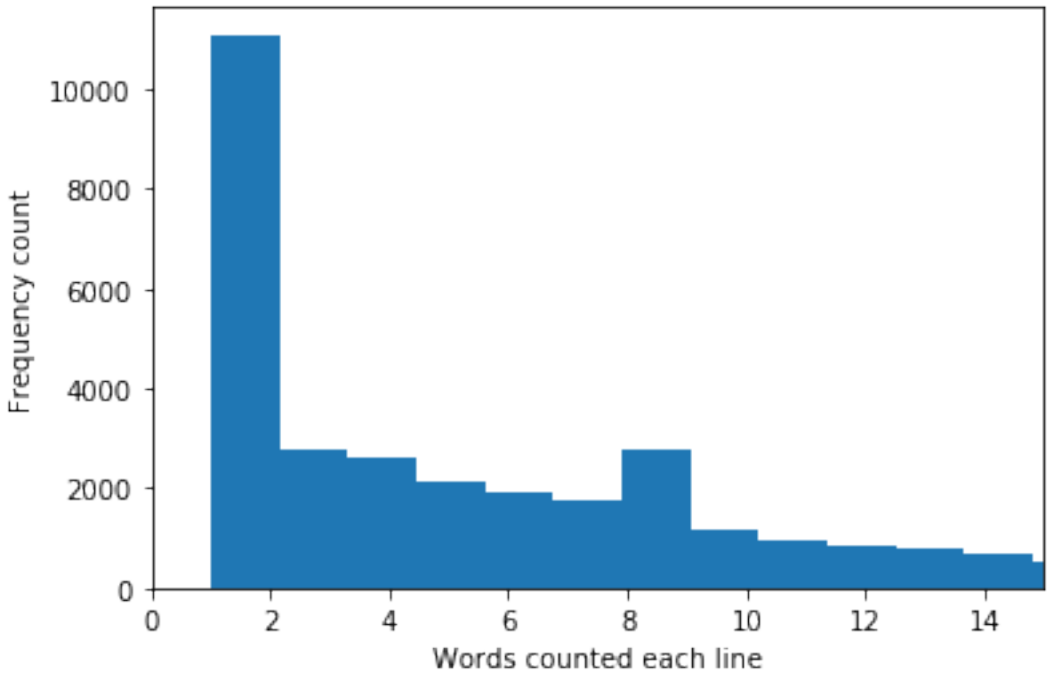


Figure 3.9: Word count per turn

In terms of the word count, as can be seen in figure 3.9 there most common value is 1. Because its proportion is significant, we do a subset of the corpus when the word_count is 1 to look closely at this part of the data. Then we look at the text content of the subgroup, including only 1 counted words. The table 3.1 below is a part of them when the frequency is higher than 100. These words could be useful as lexical cues.

Timing variables distribution

In terms of timing, when we look for all values in the whole dataset, as can be seen in Figure, the means of Latency and Overlap are 0.60 and 0.24, so generally, when there is an overlap, it is rather short, and the latency between two turns can be more evident in this corpus. This may be due to the relation and the nature of the conversation. We plot the means of each file for the Latency and Overlap columns, and we can see they are close to a Gaussian distribution.

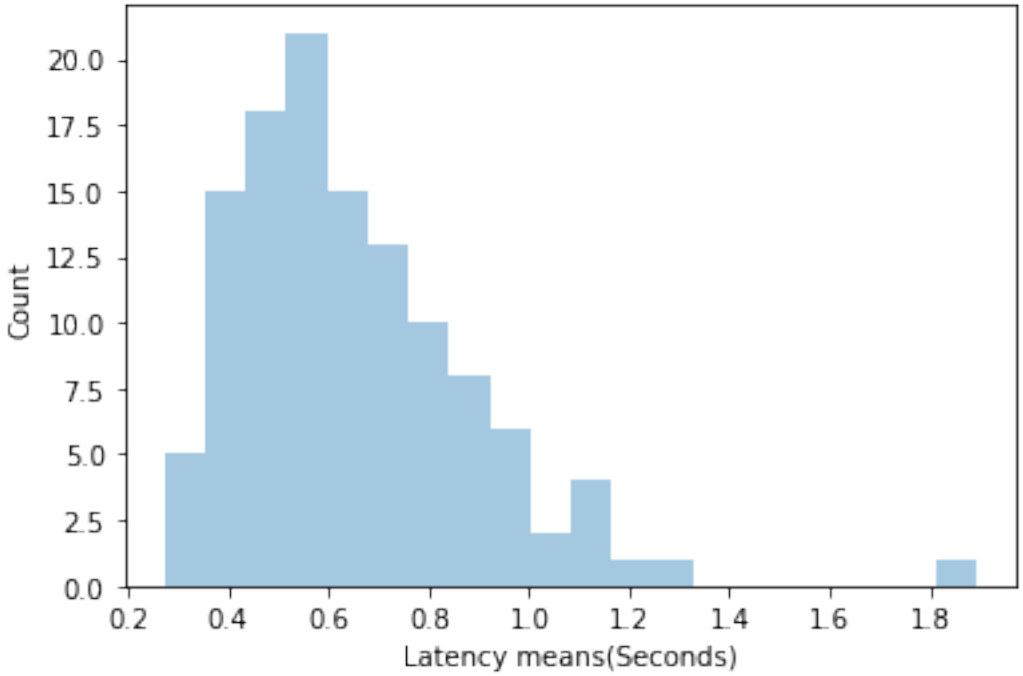


Figure 3.10: The distribution of means of Latency

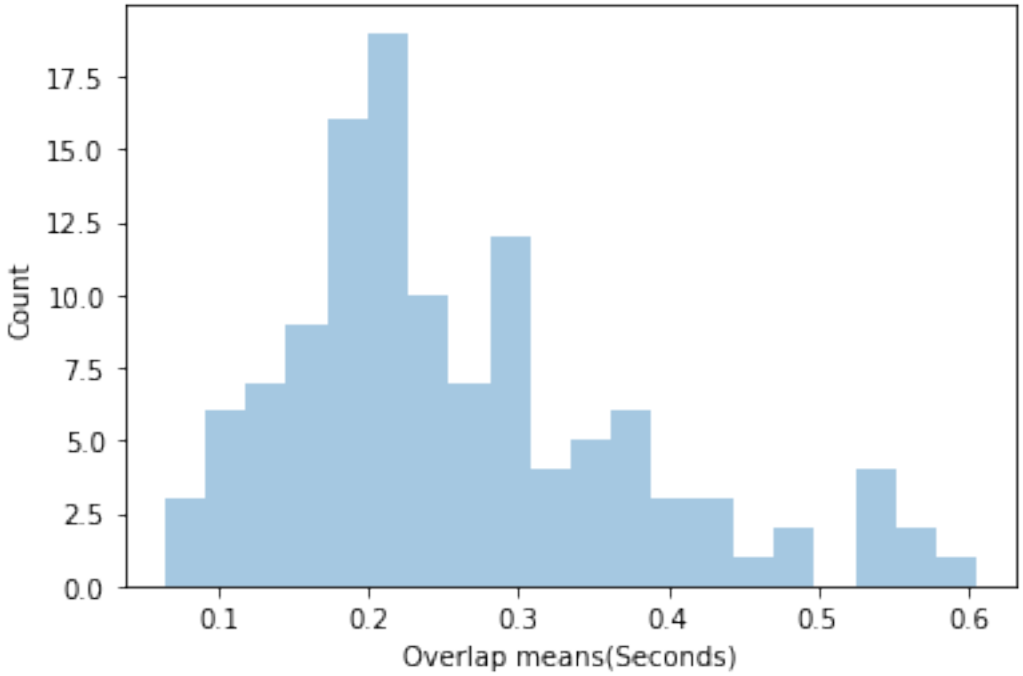


Figure 3.11: The distribution of means of Overlap

Form	Frequency	Translation
嗯.	951	Um.
{laugh}	633	{laugh}
哦.	523	Oh.
呃	474	Uh.
嗯	457	Um
啊.	398	Ah.
呃.	324	Uh.
啊?	311	Ah?
哎	310	Hey.
哦	299	Oh.
哎.	242	Hey.
对.	225	Yeah.
嗷	189	Ow
对	165	Yeah
(())	140	(())

Table 3.1: Most frequent utterances with words counted = 1.

About the Overlap and Latency in conversations, whose distributations are included in Figures 3.10 and 3.11, there are different terms and calculations in literature, sometimes the value of Overlap could be negative such as in (Heldner and Edlund, 2010). We follow the definition in (Heldner and Edlund, 2010), we use the latency term used for *"silences bounded by speech from different speakers"* , also as a lapse. Overlap defined as *"portions of speech delivered simultaneously"* in a change of floor when two participants both take part. In this study, we only consider the inter-speaker cases, so silence within one turn is not discussed. For calculation of overlap, let us state more precisely the calucation. Let us mark the start time of the utterance as S1 and end time as E1, for its antecedent, it's S and E separately. The different cases are summarized in figure 3.12.

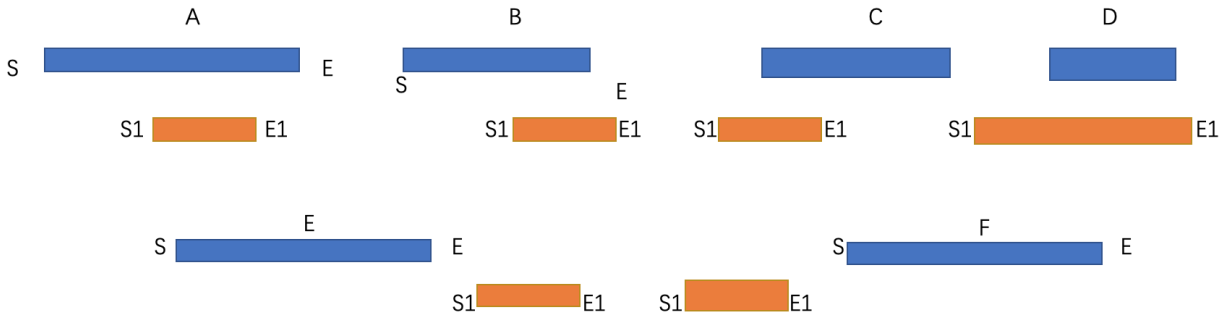


Figure 3.12: Overlap and latency cases

```
When S < S1,
    if E > E1, overlap = duration of utterance;
    if E < E1, overlap = E - S1

When S > S1,
   if E > E1, overlap = E1 - S;
   if E < E1, overlap = E- S =  duration of antecedent
```

For latency, we also only look at the cases where there is a positive value, when there is a gap between two turns, when E $<$ S1, latency $=$ S1 - E; When E1 $<$ S , latency $=$ S - E1.

Looking into specific conversation

A closer look at the data is to look at them document by document, after all the whole duration and speaker data is not fine enough as a whole.

For example, we can visualize a closer look at a specific conversation (ma_1376) through a pairplot in figure 3.13. We can tell the dominant speaker and some characteristics from this figure, seen from the x axis, like the Word_count column, we see that the speaker A produce longer utterances than B, what's more, for the duration it's also similar so generally A has the floor for the most of time. Then for the Latency, compared with B, A waits slightly less time than B when there is a blank between two turns of speech, on contrary B has a high proportion of overlap but the overlap lasts for a short time, we can infer that it's mostly backchannel. From this we can be rather sure than the A is the dominant speaker. But this kind of analysis may not apply to all the conversations because there are individual differences. Also, if the relation between the two speakers are different, like superior and subordinate, teacher and student, the proportion can be very different.

3.2.2 Annotated sub set

We selected around 5% of the whole data (the first 1500 lines of the combined data set consisting of 33485 lines) to tag the data manually to have an idea about the difference between data with NSU tags and the whole data.

The manual annotation is done by the author herself in this study for convenience and cost. In the general case, the manual annotation of the corpus should be carried by an expert annotator. In an ideal situation, there should be more than one annotator, and their results should be evaluated to get an inter-annotator agreement. The quality of the annotation could be trusted because I corrected the results several times; it took quite some time because it was unfamiliar to me at first. During the error analysis, I also corrected some mislabeled cases. The question is the easiest to determine, and there are

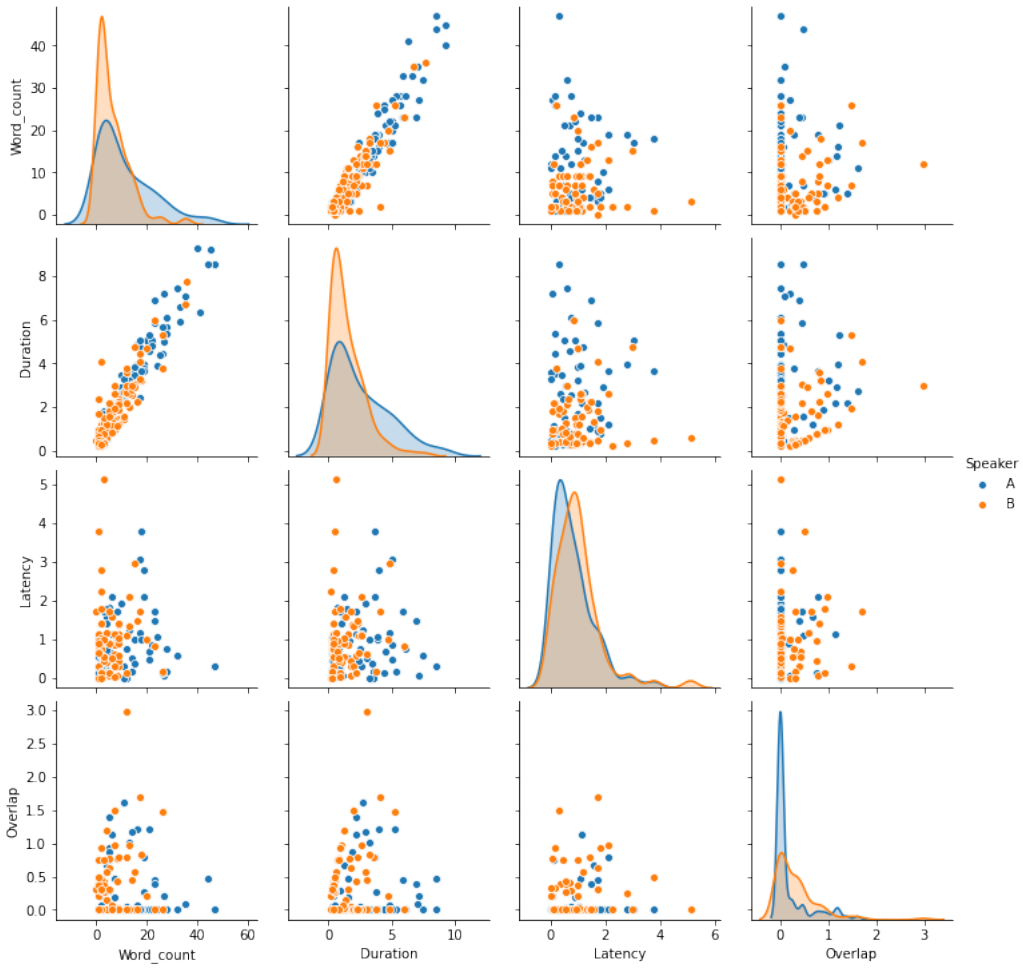


Figure 3.13: A pairplot of the file ma_1376

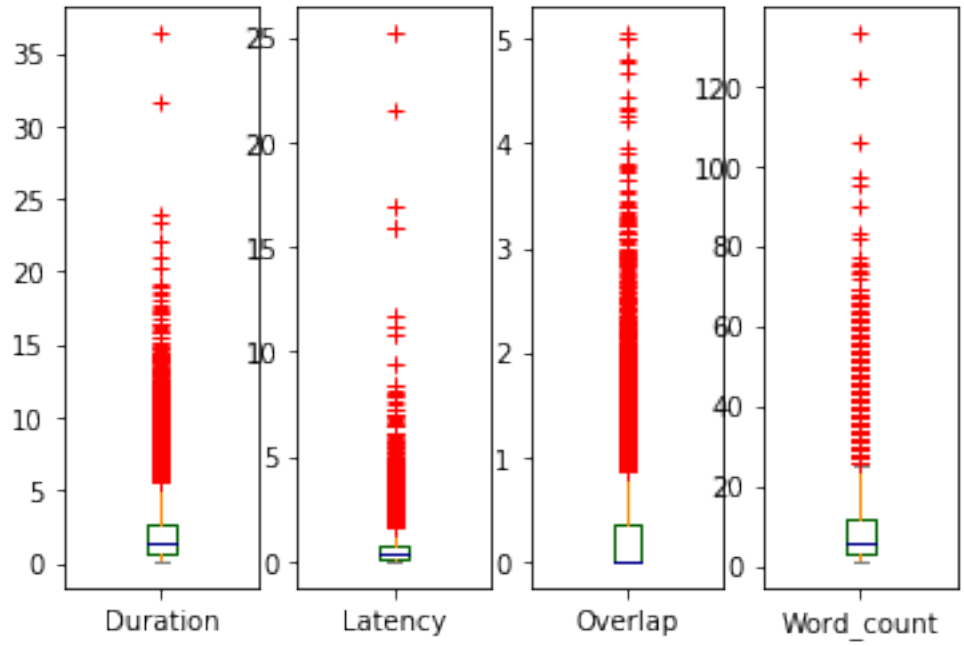


Figure 3.14: The numerical columns distribution for the whole dataset

a lot of instances repeated. The modifier classes are somewhat challenging to decide and sometimes easy to be confused with the SU.

General comparison of the variables

Because the NSUs are generally short utterances so the duration might be brief, and the counted words may be less than the mean of all the data. Same for the Latency and overlap, so we want to know the exact difference. In figures 3.14 and 3.15 are presented the results.

For the NSU labeled data, the Duration box is between 0.5 and 1, almost all below 4. The Latency box is around 0.25 to 0.7, and almost all of them are less than 4.The overlap box is from 0 to 0.5; most of them are less than 2. The Word_count column's box is from 2-4 and generally less than 8.

For the aggregate data, we can see that almost all the columns have a long tail representing a lot of outliers if we look at just the boxes. For the Duration column, it's from 1 to 3. For the Latency column, it's around 0 to 1. For the Overlap column, it's from 0 to 0.5 For the word_count column, it's from 0 to 10.

The most significant difference lies in values outside the scope of the boxes; for the Latency and the Overlap column, the box value is similar, but for the Duration and the Word_count column, the values in the tagged data are in a more limited scope.

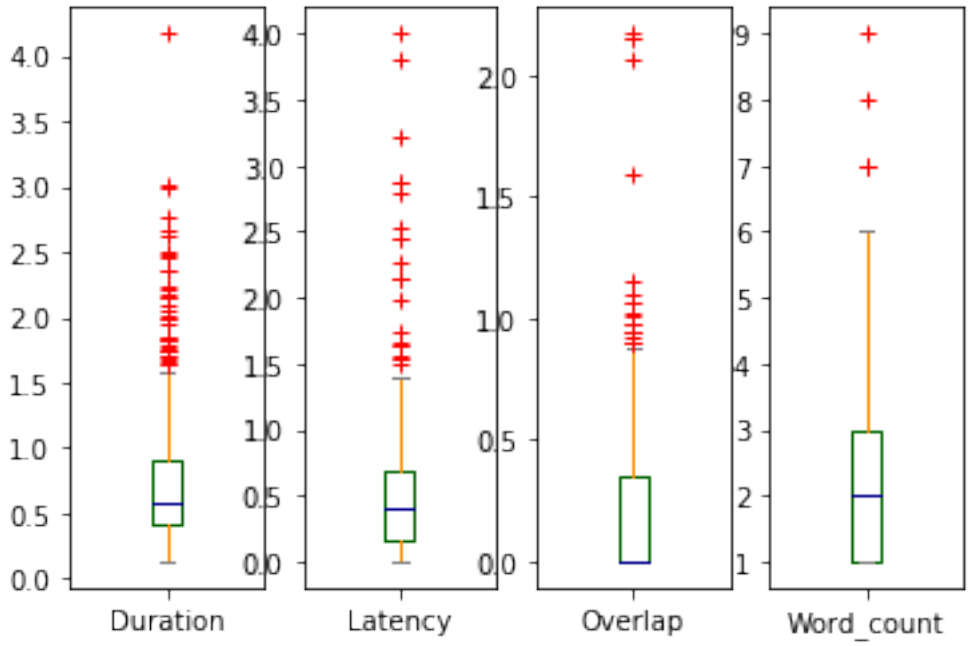


Figure 3.15: The numerical columns distribution for the sample dataset

Lexical comparison

We also take a look at the most frequent unigrams, bigrams and trigrams in the words and POS tags. Due to the segmentation, there are some words with punctuation such as %; we kept it because it's an annotation mark. Concerning words, for the unigrams, as illustrated in figure 3.16, about the difference as "I have what you don't have", we see that for SU, there is a much higher frequency of personal pronouns. For NSU labeled data, there are more final particles, such as "yah, bah, lah".

For the bigrams, for NSU data, almost all are interjection or exclamation words and their combination as demonstrated in figure 3.17. Those for expressing acknowledgment are also present, such as "yeah". In SU, there are again more personal pronouns and discourse markers such as "nage / that" "jiushi / precisely be" and "zhege/ this" .

For the trigrams, there is less overlap between the two types as we can see in figure 3.18, which can be used in extracting NSUs. But considering the counted number, they are less than the bigrams and unigrams. For the NSUs, this is similar with the bigrams, mostly interjections, and acknowledgment word, for the SU, the discourse marker "jiushishuo / that is to say" is the most frequent one, "shibushi/ yes or not" is often used in question and, "tinghaode/ nice" is a commentary expression.

'嗯'	Um	144	'我'	1SG
'啊'	ah	92	'是'	yes
'哎'	hey	89	'那'	in that c
'对'	yes	55	'的'	DE
'哦'	oh	55	'你'	2SG
'%呃'	uh	32	'个'	CLF
'%'	%	32	'就'	just
'是'	yes	32	'啊'	ah
'的'	DE	28	'了'	LE
'呃'	uh	27	'不'	NEG
'好'	good	25	'他'	3SG
'了'	LE	24	'嗯'	Um
'不'	NEG	22	'这'	this
'呀'	yah	22	'呃'	uh
'吧'	bah	14	'对'	yes
'啦'	lah	13	'说'	say
'哼'	hum	13	'哦'	oh
'那'	in that case/that	11	'一'	one
'还'	in addition	9	'哎'	hey
'钱'	money	8	'好'	good

Figure 3.16: Comparison between NSU data and the whole data for the 20 most frequent unigram word

'那', '个'	nage/that	3002	'嗯', '嗯'	um um	55
'就', '是'	jiushi/precisely be	2348	'%呃', '%'	uh %	32
'这', '个'	zhege /this	1410	'哎', '哎'	hey hey	26
'一', '个'	one	1057	'哦', '哦'	oh oh	21
'不', '是'	bushi/NEG be	914	'对', '对'	yeah yeah	17
'对', '对'	yeah yeah	875	'啊', '啊'	ah ah	14
'是', '说'	be say	795	'嗯', '哎'	um hey	13
'嗯', '嗯'	um um	613	'嗯', '哦'	um oh	11
'好', '的'	haode/ good DE	537	'哎', '嗯'	hey um	11
'我', '就'	1SG just	519	'%', '%呃'	% uh	10
'啊', '啊'	ah ah	483	'啊', '嗯'	ah um	10
'哦', '哦'	oh oh	463	'哎', '对'	hey yeah	9
'我', '我'	1SG 1SG	459	'啊', '哎'	ah hey	8
'你', '你'	2SG 2SG	458	'嗯', '哼'	um hmm	8
'是', '吧'	Yes bah	421	'的', '啊'	DE ah	8
'呃', '呃'	uh uh	415	'哎', '是'	hey yeah	8
'不', '要'	buyao / NEG need	391	'啊', '%呃'	ah uh	7
'也', '不'	too NEG	380	'呃', '呃'	uh uh	7
'的', '那'	DE that	379	'对', '啊'	yeah ah	7
'是', '不'	yes NEG	373	'对', '了'	yeah LE	7

Figure 3.17: Comparison between NSU data and the whole data for the 20 most frequent bigram words

'就', '是', '说'	jiushishuo / that is to say	712	'嗯', '嗯', '嗯'	um um um	23
'对', '对', '对'	yes yes yes	414	'哦', '哦', '哦'	oh oh oh	13
'是', '不', '是'	yes or not	261	'哎', '哎', '哎'	hey hey hey	11
'的', '那', '个'	DE nage/ DE that	212	'%呃', '%', '%呃'	uh % uh	10
'挺', '好', '的'	nice	210	'%', '%呃', '%'	% uh %	10
'个', '那', '个'	CLF nage/ CLF that	203	'啊', '%呃', '%'	ah uh %	7
'那', '个', '那'	that CLF that	201	'对', '对', '对'	yes yes yes	6
'&', '美国', '&'	& America &	176	'%呃', '%', '对'	uh % yeah	5
'我', '跟', '你'	1SG and 2SG	173	'真', '的', '啊'	really ?	5
'有', '一', '个'	there is a CLF	165	'&绿', '酶素', '&'	chloromycet	5
'是', '那', '个'	BE that CLF	162	'呃', '呃', '呃'	uh uh uh	4
'嗯', '嗯', '嗯'	um um um	152	'不', '会', '啦'	not at all	4
'哦', '哦', '哦'	oh oh oh	133	'哎', '哎', '嗯'	hey hey um	4
'那', '个', '什么'	that CLF what	126	'嗯', '哎', '哎'	um hey hey	4
'&', '中国', '&'	& China &	125	'哎', '哎', '对'	hey hey yea	4
'你', '那', '个'	2SG that CLF	124	'哎', '对', '了'	hey right	4
'个', '就', '是'	CLF jiushi	115	'嗯', '嗯', '哎'	um um hey	4
'跟', '你', '说'	with 2SG say / I'm telling you.	113	'哎', '%呃', '%'	hey uh %	4
'啊', '啊', '啊'	ah ah ah	113	'%呃', '%', '哎'	%uh % hey	4
'那', '个', '就'	that precisely	96	'哎', '嗯', '嗯'	hey um um	4

Figure 3.18: Comparison between NSU data and the whole data for the 20 most frequent trigram words

'VV'	47172	'AD'	234	'AD', 'VV'	15736	'VA', 'AD'	61
'AD'	46021	'VA'	221	'AD', 'AD'	9104	'AD', 'AD'	59
'NN'	31818	'VV'	191	'PN', 'VV'	8403	'NN', 'VV'	58
'PN'	26361	'NN'	183	'VV', 'VV'	8216	'AD', 'VA'	57
'VA'	13908	'NR'	51	'NN', 'AD'	7933	'AD', 'VV'	55
'M'	12797	'P'	47	'PN', 'AD'	7471	'VA', 'VA'	49
'DT'	10420	'CD'	37	'VV', 'PN'	7000	'NN', 'VA'	42
'P'	8495	'SP'	35	'VV', 'AD'	6743	'VV', 'NN'	37
'NR'	8448	'M'	33	'DT', 'M'	6569	'VV', 'AD'	34
'VC'	7828	'VC'	31	'VV', 'NN'	6039	'VA', 'NN'	29
'CD'	7233	'PN'	26	'NN', 'VV'	6014	'NN', 'AD'	27
'SP'	5347	'NT'	26	'CD', 'M'	5481	'CD', 'M'	26
'AS'	5317	'AS'	22	'M', 'NN'	5201	'VV', 'VV'	26
'DEC'	4406	'DEC'	17	'AD', 'VA'	5164	'VV', 'VA'	24
'NT'	3965	'FW'	14	'AD', 'VC'	4586	'VV', 'AS'	15
'VE'	3954	'DT'	13	'VV', 'AS'	4436	'VA', 'VV'	14
'DEG'	2013	'VE'	11	'AD', 'PN'	4202	'NN', 'NN'	14
'JJ'	1659	'IJ'	10	'VA', 'AD'	3407	'P', 'NN'	14
'LC'	1510	'OD'	6	'NR', 'NR'	3052	'P', 'P'	14
'CS'	482	'DEG'	6	'NN', 'PN'	2937	'VA', 'DEC'	13

Figure 3.19: Comparison between NSU data and the whole data for the 20 most frequent unigram and birams of POS tags

The correspondence of abbreviations of the tags could be found in (Xia, 2000). The unique tags on each side could be used as cues in the NSU detection task. As exhibited in figure 3.19 and in 3.20, 'JJ' refers to three types of noun-modifiers. 'LC' is localizer, 'CS' is subordinating conjunctions. 'FW' is Foreign word, 'IJ' is interjection, 'OD' is ordinal number. Likewise, other bigrams and trigrams can also be used in the detection task.

In this chapter, we looked at the weak-supervision tool Snorkel and how its workflow allows the domain expertise to maximize its use. What LFs can be used in this study and how the final model will be built. We presented the descriptive statistics of our corpus, the description of the numerical columns, and the n-gram comparison of NSU labeled data with all the data, which can be useful in later detection and classification tasks in chapters 5 and 6.

'AD', 'AD', 'VV'	3563	'AD', 'VA', 'AD'	19
'AD', 'VV', 'VV'	3363	'NN', 'VV', 'NN'	17
'PN', 'AD', 'VV'	3310	'VA', 'AD', 'VA'	17
'DT', 'M', 'NN'	2748	'AD', 'AD', 'VA'	16
'VV', 'AD', 'VV'	2547	'VV', 'NN', 'VV'	16
'NN', 'AD', 'VV'	2431	'AD', 'AD', 'VV'	16
'AD', 'VV', 'AD'	2346	'AD', 'AD', 'AD'	16
'AD', 'VV', 'PN'	2179	'VA', 'VA', 'VA'	16
'VV', 'PN', 'VV'	2058	'NN', 'VV', 'AD'	15
'CD', 'M', 'NN'	2054	'AD', 'VV', 'VV'	14
'PN', 'AD', 'AD'	1851	'VA', 'NN', 'VV'	12
'AD', 'VV', 'NN'	1778	'VA', 'NN', 'VA'	12
'PN', 'VV', 'PN'	1741	'VA', 'VA', 'AD'	12
'AD', 'VV', 'AS'	1671	'VV', 'AD', 'VV'	11
'AD', 'AD', 'AD'	1666	'NN', 'VA', 'NN'	10
'VV', 'PN', 'AD'	1648	'VA', 'AD', 'VV'	10
'NN', 'AD', 'AD'	1619	'NN', 'VA', 'AD'	10
'P', 'PN', 'VV'	1582	'VA', 'AD', 'AD'	10
'AD', 'PN', 'AD'	1467	'VV', 'AD', 'AD'	9
'VV', 'CD', 'M'	1447	'VA', 'CD', 'M'	9

Figure 3.20: Comparison between NSU data and the whole data for the 20 most frequent trigram of POS tags

Chapter 4

Mandarin Non-Sentential Utterances Qualitative Analysis

In this chapter, we will look at the NSUs in each category and examples from our corpus. Based on the literature review in chapter 2, we will see if some of the classes can be combined and on what ground.

For the whole dataset of the CallHome, there are 33485 utterances (lines) in total.

We manually label approximately 5 % of the data as a pilot study. We chose the first 1500 lines of our CallHome dataset, and according to the classification of (Wong and Ginzburg, 2018). As a result, I get 486 occurrences of NSU out of 1500 lines, which is 32.4 % in our sample corpus. We listed the categories and their occurrences below, the rest categories are below 10 in our sample.

4.1 General structure of the classification

In total, there are four big classes: acknowledgment, questions, answers, and extension moves. Those classes that are newly added compared to the original data in the taxonomy

Label code	Number of Occurrences	Label name
1	281	Plain Acknowledgement
8	67	Check Question
30	61	Disfluency
20	36	Interjection
2	31	Repeated Acknowledgement
4	16	Helpful Acknowledgement
10	15	Short Answer
6	15	Sluice
22	12	Factive Modifier

Table 4.1: List of categories and their occurences in the manually labeled data

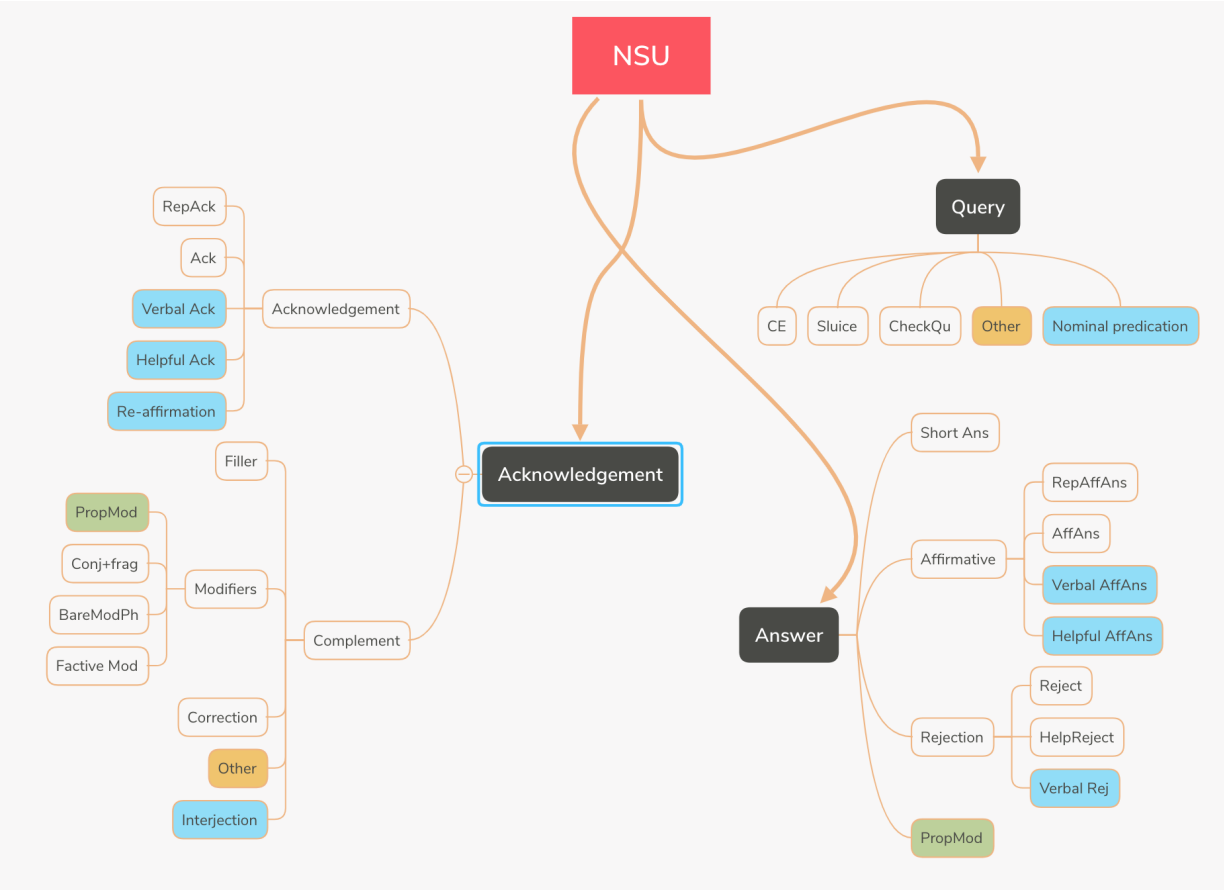


Figure 4.1: First levels and sub-classes of NSUs with added classes in (Wong,2018)

of English NSUs in (Fernández et al., 2007) are tagged with *. For each example, the designated utterance is the last one; in cases where there are three utterances, also it's the last one the NSU we refer to. The following examples mainly following the order of (Wong, 2018), as shown in table 4.2, but there are some adjustments made based on the (Fernández et al., 2007) and (Fernández and Ginzburg, 2002), for the first level categories, we decide on acknowledgment, query, answer and complement. There are some classes who doesn't fit in the ANSWERS type in (Wong, 2018), such as the INTERJECTION, and the PROPOSITIONAL MODIFIER also at two places in the original decision trees in (Fernández and Ginzburg, 2002). Thus we made these divisions; the newly added classes are integrated into the graph with blue color.

4.2 Acknowledgement

We start from the acknowledgement. We can see 4 types of Acknowledgement.

	NSU Class
	A. Acknowledgement
1	Plain Acknowledgement
2	Repeated Acknowledgement
3*	Verbal Acknowledgement
4*	Helpful Acknowledgement
	B. Questions
5	Clarification Ellipsis
6	Sluice
7*	Nominal Predication
8	Check Question
9	Filler
	C. Answers
10	Short Answer
11	Affirmative Answer
12	Repeated Affirmative Answer
13*	Verbal Affirmative Answer
14*	Helpful Affirmative Answer
15*	Re-Affirmation
16	Rejection
17*	Verbal Rejection
18	Helpful Rejection
19*	Correction
20*	Interjection
21	Propositional Modifier
	D. Extension Moves
22	Factive Modifier
23	Bare Modifier Phrase
24	Conjunction + Fragment

Table 4.2: Classification of NSU in (Wong, 2018)

4.2.1 Plain Acknowledgement

In (Fernández et al., 2007), the class PLAIN ACKNOWLEDGMENT usually has some frequent expressions like *yeah, mhm, ok*, expressing that "a previous declarative utterance was understood and/or accepted."

(24) B 呃 , 所以 他 签 的 &中国& 字 , 好 难看 哦
B e, suǒyǐ tā qiān de &zhōngguó& zì, hǎo nánkàn o, dōu méiyǒu
B Uh, so he sign DE &Chinese& characters, so ugly oh, all NEG
, 都 没有 什么 笔画 运 力 的 .
shénme bǐhuà yùnlì de.
strokes.
B Uh, so the Chinese characters he signed, they're ugly. They don't have any strokes.

A 嗯哼呃.
A en heng e.
A Em Hmm Eh.
A Uh-huh.

(ma_0882, 349.74 - 353.97)

In this example "Hm Hmm Eh." three words are used, the "嗯"(Em) is probably the most frequent word in Chinese as PLAIN ACKNOWLEDGEMENT, along with other words like "哦"(oh), "嗷"(ow), it can also combine with other words like "嗯哼"（Um-hm）, "嗯啊"(Um-ah). "呃"（uh) can also express acknowledgment, but compared with others, it's more likely to express hesitation or willingness to take the floor.

4.2.2 Repeated Acknowledgement

REPEATED ACKNOWLEDGMENT repeats a part of the previous utterance, and it is used as repeated affirmative answers.

(25) A 七百 块钱 呗 .
A qībǎi CLF money FP.
A Seven hundred money.
A Seven hundred RMB, then.

A1 七百.
A1 qībǎi.
A1 Seven hundred.
A1 Seven hundred.

(ma_1711, 570.12 - 572.22)

In this case, the part repeated is a quantifier, but it can be any type of phrase.

(26) B 哦 , 好 爽 啊 .
B ō, hǎo shuǎng a.
B Oh, how pleasant FP.
A Oh, that's good.

A 好爽, 好爽, 好爽呃.
A Hǎo shuǎng, hǎo shuǎng, hǎo shuǎng e.
A how pleasant, how pleasant, how pleasant FP.
A Great, great, that's great.

(ma_1711, 368.87 - 371.26)

In this case, we see a repeated case of Adjective Phrase（AP), and it not only repeated a part of its antecedent, but it also repeated three times.

4.2.3 Verbal Acknowledgement

* Verbal Acknowledgement, this class is defined in (Wong, 2018) as "an acknowledgement which contains just a modal verb." We didn't find any occurrences of this class in our sample corpus. So we make one to illustrate.

(27) B 健身 真的 会 让 人 变美.
B Jiànshēn zhēn de huì ràng rén biàn měi.
B Workout EMP MOD make people TRANSL beautiful.
B Workouts really make people beautiful.

A 会.
A Huì.
A MOD.
A It does.

The modal verbs in Mandarin is similar in those in English. The principal English modal verbs are can, could, may, might, must, shall, should, will and would. The can express wish and necessity and disjunction. Such as the examples in (Zhou et al., 2015): "能够、会、要、肯、敢、愿意、应该、可以（be able, will do something, want to, want to, dare to, be willing to, should, be able)" and "务必、必须、必需、必要、要、须得、得、不得不、不能不、不可不、应当、应该、理当、理应、最好、是否、可否、不妨、可以(essential, necessary, should, must, must, have to, cannot, cannot, shall, should, must, ought to, had better, if can, can, can)" etc.

4.2.4 Helpful Acknowledgment

In (Wong, 2018), * Helpful Acknowledgement differentiates from other acknowledgment classes by providing additional information.

(28) A1 // 平常 用 那个 , //
A1 // píngcháng yòng nàge, //
A1 // Usual use that, //
A1 // The usual one, //

A 是中文的.
A shì zhōngwén de.
A COP Chinese PTCL.
A It's in Chinese.

(ma_1711, 331.63 - 333.15)

In this case, A1 talks about something, and then A's utterance not only signifies that he(she) has understood what A1 was talking about, but also offers extra information about the object in question.

4.2.5 Re-Affirmation

* Re-Affirmation in (Wong, 2018) is used to affirm the authenticity of a previous utterance.

(29) B 然后 现在 就 对 我 来 说 , 就 一 钱 的
B ránhòu xiànzài jiù duì wǒ lái shuō, jiù yī qián de shì,
B then now just DIR 1SG DIR say, just one:NUM money PTCL matter,
事 , 所以 我 说 我 犹豫 啊 也 , 也 就 在 这
suǒyǐ wǒ shuō wǒ yóuyù a yě, yě jiù zài zhè dìfāng.
so 1SG say 1SG hesitate PTCL too,also just PREP this place.
地方 .

B And now, as far as I'm concerned, it's just a matter of money, so I say I'm hesitant, ah, also, right here in this place.

A 呐, 我觉得不值.
A nà, wǒ juédé bù zhí.
A nah, 1SG think NEG worth.
A Nah, I don't think it's worth it.

A 没劲.
A méijìn.
A NEG strength.
A Not worth it.

A 真的.
A zhēn de.
A really.
A really.

(ma_1711, 207.32 - 219.35)

This class is used to emphasize facts or opinions that have been stated, and it insists more on its content expressed. Besides "真的"(really) in the example, there are also other expressions like "的确"(indeed),"确实"（indeed, exactly), and so on.

4.3 Questions

There are 5 classes of Questions.

4.3.1 Clarification Ellipsis

Clarification Ellipsis (CE). In (Fernández et al., 2007), in cases when an utterance has not been fully understood, this class is used to mark the reprise fragment signaling the blank of comprehension.

(30) B 他 那个 刚 谁 讲话 啊 ? 谁 录音 的 ?
B tā nàge gāng shuí jiǎnghuà a? Shuí lùyīn de?
B He that just who speak FP? Who record FP?
B Who was that guy talking? Who recorded it?

A &大陆& 人啊.
A &dàlù& rén a.
A &Mainland& person FP.
A A mainlander.

B &大陆& 人哦?
B &dàlù& rén o?
B &Mainland& person FP.
B A mainlander.

(ma_0882,363.42 - 367.95)

The last utterance of B reuse a part of A's answer, in this case, according to the context, we can infer that A's response already cleared up B's doubts in B's first question. However, B still added a clarification ellipsis. A's answer is not a strange or rare or very technical term, so the gap of comprehension is not about the A's literal sense. We can see it as a somewhat reflective question, and A could provide more background information about his answer, or he could pass it.

4.3.2 Sluice

SLUICE. In (Fernández et al., 2007), two kinds of sluices are mentioned: reprise and direct sluices. While in the taxonomy of (Fernández and Ginzburg, 2002), reprise sluices stand apart and become CE. In (Fernández et al., 2007), "CE only includes clarification fragments that are not bare wh-phrases."

(31) A 我 在 跟 你 讲话 .
A wǒ zài gēn nǐ jiǎnghuà.
A I PREP with you talk.
A I'm talking to you.

B2 几点钟啊?
B2 jǐ diǎn zhōng a?
B2 how many clock FP?
B2 What time is it?

(ma_0717, 377.39 - 380.41)

Sluices are often in the form of wh-phrases; they are rather easy to recognize among all the categories.

4.3.3 Nominal Predication

In (Wong, 2018), * NOMINAL PREDICATION is interrogative and contains only noun phrase(s). I think this definition needs a bit modification. Like in our example from the corpus, it can include a final particle.

(32) A 昨天 , 昨天 白天 你 ((往)) 那 去 啦 ?
A Zuótiān, zuótiān báitiān nǐ ((wǎng)) nà qù la?
A Yesterday, yesterday daytime you ((to)) that go FP?
A Yesterday, yesterday, during the day, where did you go?

B 昨天白天啊?
B zuótiān báitiān a?
B Yesterday daytime FP?
B Yesterday, during the day?

(ma_0799, 559.77 - 564.13)

This is a case when the interrogative utterance contains almost only noun phrases(NP). Still, it's also easy to find that it's also a CE case because B recycles a part of A's question. Like we have said before, it's not the target's literal sense, which is unclear to B. We could see it as stalling to take some time to recall the question he should answer or a request for further information, like a more precise time of the day.

4.3.4 Check Question

Check Question. In (Fernández et al., 2007), this kind of short question is usually in conventionalized forms, such as *alright*? and *okay*?, and explicit feedback is needed.

(33) A 那 他 接 的 电话 啊 ?
A nà tā jiē de diànhuà a?
A so he pick PRT telephone FT?
A So he answered the phone?

B 啊?
B á?
B Ah?
B Huh?

(ma_0799, 585.91 - 588.3)

The "啊?"(Ah?) has very high frequency in our corpus and also in daily use, other similar forms like "哈?"(Huh?) and "嗯? "(Em ?), they are shorter than "what did you say" or " what do you mean".

4.4 Answers

Now we look at the 8 classes of Answers.

4.4.1 Short Answers

SHORT ANSWER. In (Fernández et al., 2007), this class "refers to typical responses to (possibly embedded) wh- questions. Sometimes, however, wh-questions are not explicit, as in the context of a short answer to a CE question. "

(34) A 你 现在 上 几 年级 啦 ?
A nǐ xiànzài shàng jǐ niánjí la?
A 2SG now up(DIR) which grade FP?
A What grade are you in now?

B1 三年级.
B1 sān niánjí.
B1 Three grade.
B1 Three grade.

(ma_0717, 328.51 - 330.49)

4.4.2 Plain Affirmative Answer

In (Fernández et al., 2007), PLAIN AFFIRMATIVE ANSWER is a approving answer to a polar question. Compared with PLAIN ACKNOWLEDGEMENT, it has an interrogative as antecedent.

(35) B 那 那 你 怎么 , 那 , 那个 什么 , 那 你 现在 只能
B nà nà nǐ zěnme, nà, nàgè shénme, nà nǐ xiànzài zhǐ néng dǎ yī
B so so 2SG how, so, so what, so 2SG now only CAP call one
打 一 通 电话 啊 ?
tōng diànhuà a?
CLF telephone FP?
B So, what are you, what, what, what, so, you can only make one phone call now, huh?

A 对啊.
A duì a.
A right ah.
A Right.

(ma_0082, 409.67 - 414.92)

4.4.3 Repeated Affirmative Answer

Repeated Affirmative Answer. In (Fernández et al., 2007), Repeated Affirmative Answer differentiate from the Plain Affirmative Answer by "verbatim repetition or reformulation of (a fragment of) the query. " aside from a positive response.

(36) B &绿酶素& 啊 ?
B &l ǜ méi sù& a?
B & Chloramphenicol& Huh?
B Chloramphenicol?

A 哎, &绿酶素& . 呃, 绿, &绿酶素& .
A āi, &l ǜ méi sù& . È, l ǜ , &l ǜ méi sù& .
A hey, &Chloramphenicol&. uh, Chlo-, &Chloramphenicol&.
A Hey, &Chloramphenicol& . Uh, Chlo-, &Chloramphenicol&.

(ma_0799, 426.01 - 429.41)

The affirmative answer part is "哎"(eh), which is not as evident as "yes" or "ok" and there is some disfluency like the repeated "绿"(green). Still, it's precisely an example of a Repeated Affirmative Answer.

4.4.4 Verbal Affirmative Answer

* Verbal Affirmative Answer is defined in (Wong, 2018) as an affirmative response containing only a modal verb.

(37) A 你 还 会 不会 讲 &成都& 话 ?
A nǐ hái huì bù huì jiǎng&chéngdū& huà?
A 2SG still CAP NEG CAP speak &Chengdu& words?
A Could you still speak in Chengdu dialect?

B1 会.
B1 huì.
B1 CAP.
B1 Yes I can.

(ma_0717, 284.25 - 286.7)

This one is similar to the Verbal Acknowledgement, but its antecedent is a question.

4.4.5 Helpful Affirmative Answer

* Helpful Affirmative Answer is defined in (Wong, 2018) as "affirmative response providing additional information. " It doesn't require a marker like "yes," "right," or "okay," its affirmative sense is implied with its context.

(38) B 现在 现- , 现在 的 情况 跟 原来 不 太 一样
B xiànzài xiàn- , xiànzài de qíngkuàng gēn yuánlái bù tài yīyàng.
B now now-, now PTCL situation with before NEG so same. now
. 现在 那 考试 什么 都 有人 考 , 你 知道 吗 ?
Xiànzài nà kǎoshì shénme dōu yǒurén kǎo, nǐ zhīdào ma?
that exam what all someone test,2SG know FP?
B Now, it's... it's not quite the same as it was. There's a lot of people taking the exams, you know?

A 七百块钱呗.
A qībǎi kuài qián bei.
A seven hundred yuan FP.
A Seven hundred yuan, then.

(ma_1711, 567.04 - 571.13)

In this sequence, B talks about an exam that many people take, and A makes supplementary notes about the examination fee, thus providing a piece of information that is new in the context.

4.4.6 Plain Rejection

According to (Fernández et al., 2007), Plain Rejection is not exactly the opposite of Plain Affirmative Answer, it's a negative answer, but it can respond to a not only polar query but also implicit CE question and assertions too.

(39) A 没什么 啦 ?
A méishénme la?
A NEG what FP?
A Nothing else?

B 没有了.
B méiyǒule.
B NEG FP.
B That's it.

(ma_0799, 545.78 - 546.8)

In this example, the negative discourse marker is followed by a particle "了". It's common in oral speech but rare in written text.

4.4.7 Verbal Rejection

* Verbal Rejection is defined in (Wong, 2018) as a negative answer to a polar query or an assertion, consisting of only a modal verb and a negation adverb.

(40) B1 还 要 不要 带 拖鞋 ?
B1 hái yào búyào dài tuōxié?
B1 still MOD NEG MOD take slippers?
B1 Or do you want to bring slippers?

A 不用啦.
A búyòng la.
A NEG MOD FP.
A No, it's okay.

(ma_0882, 471.62 - 473.29)

The most frequent negation adverb is "不"(no) and "没"(no) in Mandarin.

4.4.8 Helpful Rejection

Helpful Rejection. As stated in (Fernández et al., 2007), it is a negative answer to a polar question or an assertion; the context of helpful rejections can be either a polar question or an affirmation. It provides a pertinent alternative or corrects some information in the previous utterance.

(41) B 是 你们 学校 在 做 研究 吗 ?
B shì nǐmen xuéxiào zài zuò yánjiū ma?
B COP 2PL:POSS school TEMP do research FP?
B Is it your school doing research?

A 不是, 别的学校.
A búshì, bié de xuéxiào.
A NEG, other school.
A No, another school .

(ma_0882, 359.79 - 362.94)

4.5 Complement

4.5.1 Filler

As said in (Sacks et al., 1974), participants in conversation take turns to speak, provide recognizable points of possible completion of talks, and co-participants make use of the signs

to begin their discussions. When one of the speakers abandoned their turn prematurely before reaching a point syntactic and prosodic completion, the co-participant may take the floor and complete the sentence.

FILLER completes the previous unfinished utterance. In this study, we think that it should be in a situation when different speakers produce the unfinished utterance and the filler.

(42) A 对 , 我 早上 , 我 , 我 这 两天 早上 打
A duì, wǒ zǎoshang, wǒ, wǒ zhè liǎng tiān zǎoshang dǎ diànhuà, wǒ
A yes, 1SG morning, 1SG, 1SG this two day morning call telephone, 1SG
电话 , 我 曾经 哎 , 我 , 就 ((你))
céngjīng āi, wǒ, jiù ((nǐ)) [[distortion]]
once PTCL, 1SG,just((2SG)) [[distortion]]
[[distortion]]

A Yeah, I've been, I've been, I've been calling these two mornings, I've been uh, I've been, just ((you)) [[distortion]]

B 就在家里.
B jiù zài jiālǐ.
B just PREP home.
B Right here at home.

(ma_0860, 420.38 - 421.32)

B continues with the last word of A "就" (just). In this case, the A's production is very fragmented, and B completed A's phrase proactively, it may or may not fit A's intentions.

4.5.2 Correction

*CORRECTION is defined in (Wong, 2018) as the corrected version of the wrong piece of a previous utterance. Note that this class is not present in (Fernández et al., 2007) because it is included in the category "helpful rejection". And it does not involve features unique in Chinese, for example, the modal words discussed in Wong's classification.

(43) B 英文 版 最 新 的 啊 ? 英文 版 就
B yīngwén bǎn zuìxīn de a? Yīngwén bǎn jiù
B English version new:SUP PTCL FL? English version just
从 &美国& 找 啊 .
cóng &měiguó& zhǎo a.
DIR &Amercia& search PTCL.

B What's the latest English version? The English version is from the US.

A 中文版最新的啊.
A zhōngwén bǎn zuìxīn de a.
A Chinese version new:SUP PTCL FP.
A The latest in Chinese.

(ma_1711, 440.72 - 445.32)

In this example, A corrected B's question that it's not about an English version of something but its Chinese version.

4.5.3 Interjection

* In (Wong, 2018), INTERJECTION is "an abrupt expression for sudden sensations and emotions".

(44) A 就是 我们 那 届 啊 .
A jiùshì wǒmen nà jiè a.
A just 1PL:POSS that session:CLF PTCL.
A That's our session.

B 哎呦.
B āi yōu.
B EXCLAM.
B Oops.

(ma_0860, 139.28 - 140.4)

In this example, A was talking about where to go after graduation of his session. And B uses "哎呦"(Oops) to express his(her) feelings about A's speech, in this case, it's a kind of pity expressed, but using the same words or its homophone "哎哟"(ouch) or some similar words like "哎呀" (alas). Some expressions can be used to convey different emotions like regret, surprise, angry, but generally, there is more frequent use of a particular sequence for a kind of feeling.

4.5.4 Propostional Modifier

In (Fernández et al., 2007) PROPOSITIONAL MODIFIER is a propositional adverb used alone.

(45) A 你 说 我们 这儿 , 一年 就 这么 一次
A nǐ shuō wǒmen zhè'er, yī nián jiù zhème yīcì.
A 2SG say 1PL:OBJ here:LOC, one:NUM year just this:MAN one:NUM
. 你 告诉 它 一年 一次 , 它 也 不 知道
Nǐ gàosù tā yī nián yīcì, tā yě bù zhīdào.
time:CLF. 2SG tell it one:NUM year one:NUM time:CLF, it still NEG
.

know.
A You say we're here, just once a year. You tell it once a year, and it doesn't know.

B 嗯.
B ēn.
B Um.
B Um.

A 估计.
A gūjì.
A estimate.
A Possibly.

(ma_1711, 309.71 - 315.42)

The propositional modifier used here is "估计"(possibly), it modifies A's former utterance in terms of the certitude in his(her) speech.

4.5.5 Factive Modifier

In (Fernández et al., 2007), the FACTIVE MODIFIER is an evaluative adjective, also in separate use. I didn't find a case of stand-alone application of evaluative adjectives in the first 5% of our corpus. But this one below is typical.

(46) A &香港& 有 几 个 大 集团 , 在
A &xiānggǎng& yǒu jǐ gè dà jítuán, zài&xiānggǎng& ,
A &HongKong& have several CLF big group, PREP &HongKong&,
&香港& , 在 &南京& 投资 啊 ,
zài&nánjīng& tóuzī a,
PREP &Nanjing& invest PTCL,
A There are several large groups in Hong Kong, investing in Hong Kong and Nanjing,

B 嗯.
B ēn.

B Um.
B Um.

A 就是, 啊,
A jiùshì, a,
A just as, PTCL,
A Exactly,

B 小弟啊, 他这个南, &南京& 现在我告你讲, B xiǎodì a, tā zhège nán,&nánjīng& xiànzài wǒ gào nǐ jiǎng, B younger brother, he DET Nan, &Nanjing& now 1SG tell 2SG, B younger brother, he this Nan, &Nanjing& now I tell you,
A 贵的不得了,
A guì de bùdéliǎo,
A expensive PTCL amazing,
A extremely expensive,

(ma_0860, 302.01 - 309.64)

The factive modifier here is 'expensive', and in its English translation, there is an additional adverb, but "不得了" in Mandarin is an adjectival complement, and it's rather common in oral speech. If we insist on the definition, it would be "贵"(expensive). Still, this kind of single word use sounds categorical, so normally, there is an adverb to accompany the adjective, such as "好," adverb used as intensifiers.

4.5.6 Bare Modifier Phrase

Bare Modifier Phrase. In line with (Fernández et al., 2007), this class is typically PPs or AdvPs, and it refers to NSUs that act as adjuncts modifying a contextual utterance.

(47) B 后来 我 就 觉得 , 啊 ?
B hòulái wǒ jiù juédé, a?
B afterwards 1SG just think, PTCL?
B then I just thought, huh?

A 那特简单.
A nà tè jiǎndān.
A that special easy.
A That's very simple.

(ma_1711, 248.81 - 250.99)

In this case, the "特简单"(very simple) is the adjunct, with "那"(that) at the beginning referring to an exam they talked before.

4.5.7 Conjunct

In (Fernández et al., 2007),the classCONJUNCT designates fragments introduced by a conjunction.

We didn't find an example of this class in our sample corpus. In the whole corpus, some cases fulfill the requirements of this class:

(48) A 那个 总经理 , 然后 呢 ?
A nàgè zǒng jīnglǐ, ránhòu ne?
A that general manager, then FP?
A What about the general manager, then?

B 还有六个人,
B hái yǒu liù gè rén,
B and six(NUM) CLF people,
B And six more,

(ma_1711, 173.49 - 175.64)

The number of conjunctions in Chinese Mandarin is not small, but the scarcity of this kind of NSU implies that they are usually in a sentence or a clause (grammar).

4.6 Difficult cases

As we have seen in chapter 2, there are some short utterances with no verbs but do not strictly fit the standard of a certain NSU.

(49) A 哎 妈妈 啊 .
A āi māmā a.
A Eh mom FP.
A Hey mom.

(ma_0882, 189.02 - 189.73)

This is a reaction to the utterance produced by the previous speaker, which is the speaker's mother. It's most similar to class INTERJECTION, but apart from the INTERJECTION, there are other components like the name of the addressee. I hesitate to label it as an interjection or not. We propose to tolerate interjections in all categories because its semantic weight can be ignored when other constituents are present.

Besides, there also other cases, including keywords expressing surprise and other emotions with some other elements.

(50) B1 哇 这样 啊 ,
B1 wa zhèyàng a,
B1 wow that FP,
B1 Wow, that's it.

(ma_0882, 189.02 - 189.73)

This kind of expression is also widespread in conversation, "zhèyàng"(that) is a manner demonstrative.

The second confusing case is about the class PROPOSITIONAL MODIFIER, which is composed of propositional adverbs. Still, in the example below, in the last response, there is a propositional adverb "应该"(it should be that) followed by rejection, also a mixed case. This case is more complicated than the case with interjection because their semantic weight can be similar without context. One is a modifier, another is negation, so it should be decided according to the context or "pragmatic weight". For this example, the negation is more critical than the modifier, the "应该"(it should be that) is used to soften the tone of the rejection, as a euphemism.

(51) B1 还 要 不要 带 拖鞋 ?
B1 Do you still want to bring slippers?
A 不用 啦 .
A No need.
B1 啊?
B1 What?
A %啊% , 应该不用吧.
A %a% , yīnggāi bùyòng ba.
A %Ah%, should NGE MOD FP.
A
% Ah%, I don't think so.

(ma_0882, 471.62 - 476.12)

The last case is the presence of some frequent words in some utterances, like "那"(that, then) "就"(so, then), which can be omitted and doesn't influence the meaning of the utterance. In oral languages, they have high frequency; for example, in our corpus, they are among the most used uni-grams.

A 就这么荒唐.
A jiù zhème huāngtáng.

A so that ridiculous.
A It's that ridiculous.

(ma_0860, 452.58 - 453.29)

A 那还可以, 那还可以.
A nà hái kěyǐ, nà hái kěyǐ.
A that okay, that okay. A That's okay, that's not bad.

(ma_0860, 323.41 - 325.2)

4.7 Other classes

Reserved place for some classes not named yet.

4.8 Disfluencies

There are cases of disfluency we find in our corpus.

The cases of disfluency in our corpus are in the cases of (Shriberg, 1996), from the sample data with manual tag, most of them are ended with "–", so they can be deletion or substitution. We look at some examples:

Repetition:

(52) B 哎, 现在,
B Āi, xiànzài,
B Hey , now,

A 这个, 这个, 这个, (()) [[distortion]]
A Zhège, zhège, zhège, (()) [[distortion]]
A this, this, this, (()) [[distortion]]

(ma_0860, 363.80 - 365.56)

"zhège" and "nàge" can be used as discourse marker in Mandarin when they are not stressed, in (Huang, 1999) both the distal "nàge" and the proximal "zhège" are recognized as a pause marker by speakers to "make a lexical choice or to formulate a syntactic frame or to gather their thought." In (Liu, 2009), nàge/zhègee are found to serve a textual function of verbal filler. We see from this example of its repetition use.

Substitution:

(53) A 多少钱?
A duōshǎo qián?
A how much money?
A how much?

B 七百多, 七百二吧.
B qībǎi duō, qībǎi èr ba.
B Seven hundred over, Seven hundred two FP.
B Seven hundred, seven hundred and twenty.

(ma_1711, 198.57 - 202.03)

In this example, there is a substitution of words, providing a more accurate answer. It could be confused with the NSU class SHORT ANSWER, either half could enter this class, but with the self-correction, this is a disfluency.

4.8.1 Conclusion

We have looked into the NSU classes with Mandarin examples and slightly modified the first level of (Wong, 2018). For some similar NSUs, like the ACKNOWLEDGMENT and ANSWER, we mainly rely on their antecedent to decide their places. This is not exactly justified, like what we have seen in chapter 2, as pointed in (Garcia-Marchena, 2015), the classification is biased towards their places in the decision trees, once a place is decided, other alternatives will not be considered, and also because of the mutual exclusivity of the labels. About the added classes proposed in (Wong, 2018), due to their low frequency, I think in some applications, they can be integrated into their super-class without a problem. We also discuss the disfluency cases and cases challenging to decide; this could reduce the confusion when we select the NSU from the whole class, and bring some inspiration for the existing classes.

Chapter 5

Detection of NSU in our corpus

5.1 Introduction

We proceed now to our experiment. As said in chapter 3, there are two tasks, detection of NSU and classification of NSU. In practice, if we can extract the NSU from the corpus at first, this may be beneficial to the classification task. In this chapter, we will consider all the NSU as a whole compared to SU and disfluencies. We will use their common critical attributes in the Labelling Functions (LF). We will use the results of the descriptive statistics of our corpus and the in chapter 3 and the relative features in the literature, not only of NSU but also that of disfluency.

For the detection of NSUs, we refer to some criteria in (Dragone, 2015) about the extraction of NSU data from a large corpus, especially in two aspects. First, the number of words should be in a limited scope; second, "the NSU must not contain a verb contains a verb in any form." Other criteria exclude utterances containing only pauses or punctuations or unclear content, and greetings.

For the first point, in the application of our corpus, we can use the discrepancy between the labeled data in our sample corpus and the whole corpus in the numerical columns. As we have compared in the descriptive statistics, first, we use the lower-bound of the four columns: Word_count, Duration, Latency, and Overlap, to rule out the data which are impossible to be an NSU, like utterances with too many words. Second, we use the upper bound to sort the information, which is probably an NSU, as in 5.4. In this case, we only use two columns, the Duration, and the Word_count, because, for the other two columns, the difference for the labeled data and the aggregate data is rather small.

In the case of the Chinese corpus, the second criteria needs flexibility due to some verb properties in Mandarin. In English, in (Wong, 2018) it is considered that "an utterance containing no verb-predicate may be a sufficient condition for it to be an NSU," for the fact that in both traditional grammar and generative grammars a complete sentence consists

of the subject and the predicate, and the predicate is verb phrases. What's more, in the definition of NSUs given by (Ginzburg, 2012) that NSUs are "utterances without an overt predicate," but in Chinese, the predicates consist of verb predicates as well as non-verb predicates. Thus, there is some addition to the classification of NSU in Chinese.

5.2 Defining Labelling Functions

As stated before, to write labeling functions, there are several strategies: keyword matches, regular expressions, arbitrary heuristics, third-party models.

In our case, we use the first three strategies combined with three types of cues: the textual cues, the timing cues, and the contextual cues. For each kind of signal, we find the relevant features. In (Schlangen, 2005), there are two kinds of features, the structural features and the lexical/utterance-based features. We use structural features as if the two consecutive rows of utterances belong to the same speaker. The lexical-utterance based features include the presence of (tensed) verb(it didn't discuss the Chinese Mandarin), the presence of disfluency, presence of question mark in the antecedent, presence of wh-word in the precursor, the ratio of polar particles in the antecedent, length of the utterance, length of its antecedent, the proportion of noun/non-nouns, of same nouns, of identical words in the utterance and its forerunner, and at last, the google similarity in (Cilibrasi and Vitanyi, 2007). It's a normalized semantic distance based on the search results of the search engine google. . We also apply some of them in our experiment. In (Fernández et al., 2007), as mentioned in chapter 2, there are three sets of features: NSU features, Antecedent features and Similarity features. The whole set of features includes judgments of proposition or question about an utterance, presence of wh_word, yes/no word, lexical items such as the modifiers, and the conjunctions in an utterance. Regarding structural features, it's about the antecedent. In our study, the features are related to these points:

1. Is the utterance a question
2. Presence of a wh_word in the utterance
3. Presence of yes/no word in the utterance
4. Presence of words of acknowledgment
5. Presence of words of different lexical items representing factive modifier, propositional modifier, etc.
6. Whether the same speaker produces the two consecutive utterances
7. Whether the antecedent is unfinished
8. Number of same words in the NSU and the antecedent

```
@labeling_function()
def duration(x):
    return SU if x['Duration']  > 4  else ABSTAIN

@labeling_function()
def latency(x):
    return SU if x['Latency']  > 4 else ABSTAIN


@labeling_function()
def overlap(x):
    return SU if x['Overlap']  > 2 else ABSTAIN
```

Figure 5.1: Size-related LF for detecting NSU

9. Number of identical tags in the utterance and its antecedent

We compared the features in (Fernández et al., 2007) and (Schlangen, 2005) to find the shared essential elements and also based on the characteristics of our corpus. Such as we don't use the google similarity. These features are the most basic ones; others may be tested yet not necessarily useful. These features are embodied in the LFs for different kinds of NSUs. They can be combined with some numerical indexes, such as the duration or length of utterance.

5.2.1 Keywords-based Labelling Functions

First, for the keywords, we create a set of lexicons for each category of NSUs, identifying the presence of words/patterns for the features 2,3,4,5 listed above, this whole set of lexicon serves as a dictionary. With the dictionary and the limits of the Duration and Word_count, we aim to capture these NSUs with some confidence: Plain Acknowledgement, Sluice,Plain Affirmative Answer,Plain Rejection, less precise but could be useful for the classes concerning modifiers. The realization in LF is to search the defined keywords in the utterance, as in figure 5.2.

5.2.2 Size-related Labelling Functions

Also, as we have done in the data exploration, we choose from the results of the "word_count" = 1, which will cover most of the frequent words/patterns in NSU to make sure they are included sorting the NSU out. Some persistent final particles can also be used for proposi-

```
@labeling_function()
def nsufreqw(x):
    return NSU if x['Text'] in["嗯 ,","嗯 ?","啊 ?","啊哈 .",
                               "哎 ,","哎 .","什么 ?"
                               "对 .","喔 .","哎 ?","呃 .","呃 ,","哈 ?",
                               "哦 .","哦 ," ] else ABSTAIN
```

Figure 5.2: Lexical-related LF for detecting NSU

```
@labeling_function()
def wordcount(x):
    return SU if x['Word_count']  > 7 else ABSTAIN

@labeling_function()
def ywordcount(x):
    return NSU if x['Word_count']  in [1,2,3] else ABSTAIN
```

Figure 5.3: Size-related LF for detecting NSU and SU

tion/question judgment for feature 1. This could be useful to find the query-related NSUs. We use the size-related limitations for utterances tend to be NSU or SU like in figure 5.3

5.2.3 Timing-related Labelling Functions

Regarding timing cues, apart from the upper limit and lower limit for NSU categories as a whole for the detection task as shown in figure 5.4, an internal distinction could be made among classes for classification tasks in the next chapter. For example, some generally short classes like PLAIN ACKNOWLEDGMENT compared with some more extended types such as

```
@labeling_function()
def duration(x):
    return SU if x['Duration']  > 4  else ABSTAIN

@labeling_function()
def yduration(x):
    return NSU if x['Duration']  < 0.8  else ABSTAIN
```

Figure 5.4: Size-related LF for detecting NSU and SU

```
@labeling_function()

def has_speaker(x):
    """If several continuous utterances is produced by the same speaker, then it's not a feedback"""
    if x['Same_speaker'] == True:
        return SU
    return ABSTAIN
```

Figure 5.5: Size-related LF for detecting NSU and SU

```
@labeling_function()

def pos2(x):
    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    for tag in tags:
        if tag in ['JJ','LC','CS' ]:
            return SU
    return ABSTAIN
```

Figure 5.6: Using unique POS tags for SU detection

HELPFUL REJECTION, the duration column information is valid. For a more nuanced distinction between classes, the "overlap" or "latency" can be beneficial.

5.2.4 Context-related Labelling Functions

For contextual indices, we rely on the rest of features in LFs to clarify the relations between the utterance and its forerunner. For example, if the same speaker produces two consecutive utterances, the latter is probably not an acknowledgment as shown in figure 5.5. If the preceding utterance is an unfinished production, then the utterance can be a FILLER. For the 8th and 9th features, it helps find the classes implying semantic relation, such as CLARIFICATION ELLIPSIS, REPEATED AFFIRMATIVE ANSWER and REPEATED ACKNOWLEDGMENT.

5.2.5 Exploiting POS-tagging

The POS-tags can be used in selecting single NSUs but also very helpful in contextual connection. We also choose the distinguishing features of NSU compared with Sentential Utterances and Disfluency. We compared the unigrams, bigrams, and trigrams of tags in terms of their occurrences and proportions. We choose the tags which are more likely to be in the NSUs than in other categories. For the relation, we search and count the repeated words/tags between the two neighboring utterances. The example in figure 5.6 uses the results of the POS tag comparison between NSU and SU in the data

5.3 Snorkel modelling

The decision in the detection task is whether or not it's an NSU, we assign three labels for yes, no and abstain. We use the distinctive features of NSU in terms of its Duration and number of words in an utterance. The disfluency and other short utterances may be difficult to separate, we tried the related labeling functions but they did not enter the final LF set, the decision is based on their coverage of the dataset, their conflicts with other LF and their accuracy. We also have LF about the frequent words to make sure the probable NSU are included.

In Snorkel, we see the label frequency in the train dataset that over half of our train dataset data points have two or fewer labels from LFs. The baseline is taking the majority vote for each data point; each LF will cover a portion of data. The MajoritLabelVoter of Snorkel's inadequacy is that each vote of the LF are considered of equal efficiency, but this is not the case. Snorkel's LabelModel deals with the correlation among LFs when combining all the outputs of the LFs. We get a Label Model Accuracy of 80.4% compared with the Majority Vote Accuracy: 83.1%. The Label Model Accuracy is not always higher than the Majority Vote Accuracy. [1] From the section 3.1 in (Ratner et al., 2017b), it's explained that for very sparse label matrices (almost no conflicts among LFs) or very dense label matrices (a lot of conflicts among LFs) will probably lead to this result. As we have tested different LFs and their performances, we had two versions of notebook, one with a final set of LFs with accuracy higher than 75%, and 5 out of 8 LFs have accuracy over 94%, and another notebook with 13 LFs and with a wider range of accuracy(66% - 1). Their internal conflicts are also different; the former ones with fewer LFs have fewer conflicts among the LFs, and only two have conflicts over 0.1, while the latter has conflicts to 0.4. For these two versions of notebook, their Label Model performances are slightly lower than the Majority Model.

5.4 Classification Method and Results

After trials with different LFs, we decide on a final set of LFs and their performance is in figure 5.7.

The last step is to feed the results of the LabelModel of Snorkel in this experiment to the Keras and Scikit-Learn classifiers. In Keras we get a test accuracy of 79.3% and Scikit-Learn a test accuracy of 76.3%.

[1] See discussion in the forum: https://spectrum.chat/snorkel/help/is-label-model-always-bettern-than-majority-model 184c2bcf-ef8f-4b7f-93b0-4a0d77e336b5

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
duration	0	[0]	0.095556	0.095556	0.004444	86	0	1.000000
wordcount	1	[0]	0.335556	0.335556	0.046667	301	1	0.996689
yduration	2	[1]	0.296667	0.296667	0.151111	208	59	0.779026
ywordcount	3	[1]	0.366667	0.366667	0.197778	251	79	0.760606
english	4	[0]	0.016667	0.016667	0.016667	14	1	0.933333
wordcount_disfluency	5	[0]	0.271111	0.271111	0.228889	160	84	0.655738
nsufreqw	6	[1]	0.060000	0.060000	0.024444	52	2	0.962963
pos2	7	[0]	0.120000	0.120000	0.022222	105	3	0.972222
pos_heavy	8	[0, 1]	1.000000	1.000000	0.442222	769	131	0.854444
pos_light	9	[0, 1]	0.751111	0.751111	0.314444	519	157	0.767751
pos_verb	10	[0, 1]	1.000000	1.000000	0.442222	682	218	0.757778
has_speaker	11	[0]	0.194444	0.194444	0.098889	139	36	0.794286
keyword_就	12	[0]	0.290000	0.290000	0.072222	250	11	0.957854

Figure 5.7: Results of the detection task with its LF set

5.5 Error Analysis

The final set of LFs covered all the data with the LF pos_heavy. The other LFs are way behind its coverage. So for the whole data set, all the data points have received a label. But there is always a question of balance between coverage and accuracy, we have tried with LFs with high coverage but with an accuracy around 78% and also with less satisfying results in Keras and Scikit-Learn classifiers.

We have discovered some points such as the word-POS pairs usually in SU, from the error analysis done gradually for each small group of LFs before the final set. Disfluency is an obstacle for the NSU because they have many similarities in Duration and word_count. If we don't use the punctuation, some truncated cases, those disfluencies without repetition and repair, and purely false start then abandoned, it's hard to describe them with LFs if we don't use the punctuation.

In this chapter, we have seen how we use LFs for the detection of NSU; we first introduced the set of features that could be integrated into the LFs. We showed the LFs in detail, the keywords-based, size-related, timing-related, context-related examples, and the POS-tag implication. We then build a model and see its performance. We did an error analysis, hoping to record some lessons gained.

For the classification task, it will be practical to group some categories. Otherwise, our LFs may be too sparse. We will present our choice of classes (those in the majority in number) and how we did the modeling for the chosen categories.

Chapter 6

Classification

6.1 Introduction

Compared with the detection task, the classification task is more complex. First, because the attributed labels are a lot more than the detection task, second point is the significant imbalance of the labels. The ideal would be to classify all the tags with accuracy, but in this study, we didn't achieve it. First, our available clues about these data are restricted. The features we find may be insufficient to describe the data. Second, inside one big category, such as acknowledgment, the similarity between the subcategories makes it hard to separate them. We managed to classify some groups, though.

For a limited classification task, we have two choices on deciding which classes to include. One is the "majority" classes based on the manual annotation results; we identified the categories with a higher frequency in the dataset(the types who have a proportion higher than 3% among the data labeled as NSU). These four categories are Plain Acknowledgement, Repeated Acknowledgement, Check Question, and Interjection. Another way is to use the first-level categories: acknowledgment, question, answer, and extension moves.

6.1.1 Multi-class labelling

The first thing we want to elucidate is about some terms used in machine-learning. What kind of classification? Is our task a Multiclass, or a multilabel problem? According to the scikit-learn 0.22.2 documentation [1], in a multilabel classification task, for the data points there are x labels from n classes, there are several properties, and each property can have attributes more complex than binary. It is about assigning a data point with non mutually exclusive label(s). The classification principle in (Schlangen, 2003) fit this. The mutual exclusivity of labels is the essential difference with Multiclass classification, which assumes that each sample is assigned to one and only one label. For example, one sample can not be an animal and a plant simultaneously. Therefore, our NSU classification task is a

[1] https://scikit-learn.org/stable/modules/multiclass.html

multiclass classification problem, because each fragment can only be attributed with one and only one label of NSU.

6.2 Simplified taxonomy classification

We first try with the first choice: PLAIN ACKNOWLEDGMENT, CHECK QUESTION, and INTERJECTION, using LFs combining the cues. The keywords and POS tags are chosen to sort out the corresponding classes, considering the potential conflicts among classes. The timing-related cues are not used in this task because the duration is all generally concise for the three classes except for the REPEATED ACKNOWLEDGEMENT. For this class, the duration distribution is also not stable. Thus no rational limits could be set in this case.

6.2.1 Keywords-based LF combined with size-related LF

For PLAIN ACKNOWLEDGMENT, as most of them are counted as one word, and they are declarative utterances, we use a regular expression to label the statements with less than two counted words and end with comma or period. We use another LF to include frequent words. Apart from the keyword match, we also combine the use of regular expressions in labeling functions to match words at a limited location, like the question mark at the end of an utterance. With the same combination of words, the presence of a question mark can change the classification of an NSU. For example, from PLAIN ACKNOWLEDGMENT (嗯. Um .) to CHECK QUESTION(嗯? Um ?). Despite the intersection, we choose the words which are more likely to appear in a category than in another. Thus we get "嗯呃哦嗷哎" as the keywords in PLAIN ACKNOWLEDGMENT .

For the CHECK QUESTION and INTERJECTION, we also follow the same principle and write two LFs for each representing their characteristics in keywords and timing-related indices, as shown in figure 6.1.

6.2.2 Exploiting POS-tagging

About the POS tag information, we use some LF to describe the NSUs with unigram, bigram and trigram separately. We made a subset of these NSUs and observed what n-grams could separate them from other NSUs. For the PLAIN ACKNOWLEDGEMEN, we use the unigram, bigram and trigram at the first trial as in figures 6.2 and 6.3 because this class has a high frequency, but we can envision a conflict and overlap with other classes.

We did the same with CHECK QUESTION and INTERJECTION to get a set of LFs in this task.

6.2.3 Context-related Labelling Functions

The class REPEATED ACKNOWLEDGMENT doesn't have fixed word patterns; its content is highly related to its antecedent, so we can only rely on the contextual index, we wish

```
@labeling_function()

def lf_ackplain(x):
    if x['Word_count'] < 2:
        if re.search(r"[嗯呃哦嗷哎]", x.Text, flags=re.I) :
            return ACKPLAIN
    return ABSTAIN
@labeling_function()

def lf_checkq(x):
    if x['Word_count'] < 2:
        if re.search(r"[啊吗嘛]", x.Text, flags=re.I) :
            return CHECK
    return ABSTAIN
@labeling_function()

def lf_interj(x):
    if x['Word_count'] < 3 :
        if re.search(r"[呀哼哇]", x.Text, flags=re.I) :
            return INTERJ
    return ABSTAIN
```

Figure 6.1: Example of LF combining timing cues and textual cues

```
@labeling_function()


def ackplain_pos(x):
    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    count = 0
    for tag in tags:
        count +=1
    if x['Word_count'] < 2:
        if count < 4:
            if tag in ['NN','AD','VA','PU']:
                return ACKPLAIN
    return ABSTAIN
```

Figure 6.2: Example of LF using unigram POS cues, PLAIN ACKNOWLEDGEMENT

```
@labeling_function()
def ackplain_posbi(x):

    bi_tags = nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    bi_tags = list(nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()]))
    if x['Word_count'] < 2:
        if (('AD','PU') in bi_tags):
            return ACKPLAIN
    return ABSTAIN
@labeling_function()
def ackplain_postri(x):

    tri_tags = nltk.trigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    tri_tags = list(nltk.trigrams([utt.split('_')[1] for utt in x['Tagged'].split()]))
    if x['Word_count'] < 2:
        if ((('NN','VV','PU') in tri_tags) or (('NN','VA','PU') in tri_tags)) :
            return ACKPLAIN
    return ABSTAIN
```

Figure 6.3: Example of LF using bigram and trigram POS cues, PLAIN ACKNOWLEDGEMENT

to know the similarity between two neighboring utterances. We aim at getting a number of shared values and their ratio compared to the length of the utterance. We added a column to get the repeated POS tagged transcription portion of the utterance and its precursor. To capture this class, by limiting the number of words in an utterance, if there is a high proportion of shared words with its antecedent, then it's probably REPEATED ACKNOWLEDGEMENT, demonstrated in figure 6.4.

We also have a LF for SU with the assumption that if the common words between the utterance and its antecedent is less than 50%, then it's not NSU.

6.3 Classification Method and Results

For our model of classification for major classes, the result of Majority Vote Accuracy is 72.0%, the Label Model Accuracy is 72.3%. The result is a F1 score, as explained in (Sasaki, 2007), it is Micro average for the multiclass setting, this "calculates metrics globally across classes, by counting the total true positives, false negatives and false positives", we get our f1_micro of 0.74. And the test accuracy of this Scikit-learn classifier is 74.7%.

For comparison, we use the result of another classification as the baseline. The compared classification is for the first-level categories, [2] they are QUESTIONS, ANSWERS, ACKNOWLEDGEMENT and COMPLEMENTS.

For this classification model of first-level categories, the result of Majority Vote Accuracy is 65.7%, that of Label Model Accuracy is 66.0%. The F1 micro is 0.91. The test accuracy of the Scikit-learn classifier is 74.3%.

[2] As the within-category shared more consistency, the results look better. As we have the same logic of LFs and this classification is easier, we don't present the details here; please feel free to read the corresponding notebook in the annex.

```
@labeling_function()


def reack(x):
    '''By limiting the number of words in an utterance, if there
    words = [utt.split('_')[0] for utt in x['word_overlap']]
    count = 0
    for word in words:
        count +=1
    if x['Word_count'] < 6:
        if count/float(x['Word_count']) > 0.8:
            return REACK
    return ABSTAIN
```

Figure 6.4: Example of LF of contextual cues used for REPEATED ACKNOWLEDGEMENT

We also attempted to classify all the NSUs, adding LFs in a similar vein as the other classification done for major classes. Still, I haven't come up with LFs for BARE MODIFIER PHRASE and CORRECTION who are extremely hard to capture. The results for this LF set is presented in figure 6.5. And also FILLER if we are not going to use the punctuation as an index. We can use features like size-related LFs to detect disfluency like unfinished sentences, but we cannot use the disfluency as a given condition in the notebook to detect FILLER. With a number of zero coverage for some LFs, the model produced has Majority Vote Accuracy of 54.0% and Label Model Accuracy of 53.7%. The F1 micro is 0.86 and the Test Accuracy of Scikit-learn classifier is 67.7%.

6.4 Error Analysis

The table of all the LFs and the results are below in figure 6.6:

We see that the LF to capture the classREPEATED ACKNOWLEDGEMENT is far from expected and even with changes in the number of the LFs, the result doesn't become better.

We used the helper method get_label_buckets(...) to find the indices of data points that the LF labeled 'ACKPLAIN' that THE SYSTEM THINK belongs to class 'CHECK'. We see a lot of them did have a big enough portion of repetition, but they are short sentences because their predicate is saturated or more directly; they contain verb predicate, and these verbs are not modal verbs as shown in figure 6.7.

Another error analysis is done with the data points that the corresponding LF labeled 'ACKPLAIN' that THE SYSTEM THINK actually belong to class 'CHECK'. From the results we get we see most of them are cases with "啊"(ah), but they are not final particles with question mark, as we don't use punctuation marks as index. A lot of check questions use "啊"(ah), but "啊" (ah) is also commonly used in plain acknowledgement. Yet as we have said before, we avoid as we can to use same keywords in LFs for different classes, and

lf_sluice	14	[6]	0.006645	0.003322	0.003322	1	1	0.500000
lf_ce	15	[5]	0.102990	0.083056	0.083056	2	29	0.064516
lf_ackhelp	16	[4]	0.023256	0.023256	0.023256	1	6	0.142857
pos_interj	17	[20]	0.023256	0.023256	0.023256	0	7	0.000000
pos_nompre	18	[]	0.000000	0.000000	0.000000	0	0	0.000000
short_ans	19	[10]	0.009967	0.009967	0.009967	0	3	0.000000
aff_ans	20	[]	0.000000	0.000000	0.000000	0	0	0.000000
ans_vaff	21	[]	0.000000	0.000000	0.000000	0	0	0.000000
ans_raff	22	[]	0.000000	0.000000	0.000000	0	0	0.000000
ans_rej	23	[]	0.000000	0.000000	0.000000	0	0	0.000000
ans_vrej	24	[]	0.000000	0.000000	0.000000	0	0	0.000000
ans_propmod	25	[]	0.000000	0.000000	0.000000	0	0	0.000000
pos_facmod	26	[22]	0.003322	0.003322	0.003322	0	1	0.000000

Figure 6.5: Results of the newly added LF performance compared with the four classes, many have zero coverage, and some only have incorrect instances.

from the sample data we see 啊"(ah) as a single words is much more frequent in CHECK QUESTION, we used this word for the class CHECK QUESTION. It's almost impossible to differentiate them without punctuation.

Another (maybe not so important) problem is in the 'Text' column they are not question yet in the 'Tagged_version' there is question mark, a bit bizarre.

6.5 Discussion

The proportion of NSU in our sample corpus is 32.4%, which is extra-high compared to the results of around 10% in (Fernández et al., 2007)(also in other studies like (Fernández and Ginzburg, 2002),(Schlangen, 2003),(Schlangen, 2005)) mentioned in chapter 1. There are several possible reasons, the feedback/backchannels are ubiquitous in the class PLAIN ACKNOWLEDGMENT such as "嗯" and also in the CHECK QUESTION as "啊？ ", we see in (Jurafsky et al., 1997) the proportion of backchannel reaches 19% in an SWBD conversation corpus study. Also, due to the nature of our corpus, because the conversation parties can't see each other, the listener tends to provide more phrasal backchannels to show his(her) attention. It is compared in (Harvey, 2011) a call center corpus and the Switchboard corpus, the backchannel frequency in these corpora are higher than the face-to-face conversation. Because the interlocutors can't see each other through the telephone, they tend to produce more backchannels to keep the talk flow uninterrupted and assure the dominant speaker.

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
duration	0	[0]	0.107778	0.107778	0.000000	97	0	1.000000
wordcount	1	[0]	0.407778	0.371111	0.000000	360	7	0.980926
has_speaker	2	[0]	0.188889	0.166667	0.040000	124	46	0.729412
lf_ackplain	3	[1]	0.104444	0.104444	0.058889	89	5	0.946809
lf_checkq	4	[8]	0.036667	0.036667	0.036667	15	18	0.454545
lf_interj	5	[20]	0.021111	0.017778	0.013333	15	4	0.789474
ackplain_pos	6	[1]	0.181111	0.170000	0.124444	107	56	0.656442
ackplain_posbi	7	[1]	0.056667	0.056667	0.035556	50	1	0.980392
ackplain_postri	8	[1]	0.012222	0.012222	0.003333	8	3	0.727273
check_posbi	9	[8]	0.002222	0.002222	0.002222	1	1	0.500000
interj_posbi	10	[20]	0.014444	0.008889	0.004444	7	6	0.538462
reack	11	[2]	0.127778	0.117778	0.117778	5	110	0.043478
su_overlap	12	[0]	0.564444	0.400000	0.018889	421	87	0.828740
pos_dis	13	[0]	0.022222	0.010000	0.000000	13	7	0.650000

Figure 6.6: Results of the LFs set for the main classes

Tagged	add	word_overlap
哎_NT ,_PU 你_PN 在_P 记_NN ._PU 呃 _VA ._PU	我_PN 在_P 记_NN ._PU	{._PU, 记_NN, 在_P}
哎_AD ,_PU 我_PN ,_PU 我_PN 自己_PN 用_VV ._PU	我_PN 自己_PN 也_AD 在_AD 用_VV ._PU	{._PU, 我_PN, 自己_PN, 用 _VV}
哎_NT ,_PU 我_PN 知道_VV 了_AS ._PU	我_PN 的_DEG 情况 _NN ,_PU 你_PN 晓得 _VV 了_AS 吧_NN ? _PU	{,_PU, 了_AS, 我_PN}
哦_AD ,_PU 我_PN 没_AD 听_VV 清楚 _VA ._PU	我_PN 两_CD 件_M 事 _NN 都_AD 说_VV 了 _AS ,_PU 红白_JJ 喜...	{._PU, ,_PU, 我_PN}
还_AD 没_AD 结_VV 张_NR 了_AS ,_PU	还_AD 没_VV 结_VV 张 _M ,_PU	{结_VV, ,_PU, 还_AD}
我_PN 一直_AD 在_P 家_NN 哎_VV ,_PU 啊_SP ?_PU	那_DT 个_M 电话_NN 我们_PN 通过_P 来 _VV 了_AS ,_PU 你 PN	{,_PU, 啊_SP, ?_PU}

Figure 6.7: Results of mislabeled PLAIN ACKNOWLEDGEMENT

One difficulty in classifying the NSU is that it's context-dependent. To determine the label of a fragment, we need to look at its antecedent. According to (Schlangen, 2005), there are three reasons why it's hard when to consider the antecedent: First, it is not always the previous utterance that provides the directly related information, in their data, the average distance is 2.5 utterances. Second, it's not necessarily a single utterance that does this–it might very well be a span of statements or some implied information from such spans.

Another question we need to consider is the imbalance of labels in the dataset. When we did the classification for all the tags, as we test the coverage with the development set, there is zero coverage let alone the accuracy, conflicts or other indicators. So their contribution in the final set of LFs is unknown.

For cases like Propositional Modifier and Factual Modifier, their syntactic form is very clear but when we use relative POS tag combination to set them in a LF, there are some problems. In the sample corpus, there are no case exactly corresponds to a stand-alone factual modifier adjective, because there can be some final particles such as "哎","啊" or other patterns. In sum, they are not as "clean" as in the definition. Set aside this detail, we can set a collection including the POS tags matching the type of modifier. But depends on the tool of tagging, there are some misalignment, some polymeric words may be tagged wrongly, like "可以", it can be used as a verb, expressing modal verb, but also it can be used as an adjective , such as "not bad, nice", similar to "不错". Also, the same words with same meaning in the same utterance can be tagged differently :

(54) 呃_AD ,_PU 那_DT 蛮_NN 好_AD 蛮好_VV ,_PU
e_AD,_PU nà_DT mán_NN hǎo_AD mán hǎo_VV,_PU
uh_AD,_PU that_DT quite_NN good_AD quite good_VV,_PU

(ma_0963, 305.12 - 307.63)

哦_AD ,_PU 厉害_NN 厉害_VA ,_PU
Ó_AD,_PU lìhài_NN lìhài_VA,_PU
oh_AD ,_PU great_NN great_VA ,_PU

(ma_0963, 319.64 - 320.46)

In the first example above, the "蛮好"(quite good) were segmented and received different tags, in the second example, the "厉害" appeared twice but received different tags. So this may influence the result of our LF.

Chapter 7

Conclusion

In this dissertation, we present our work regarding non-sentential utterances. NSUs are utterances partial syntactically but convey integral meaning semantically. We chose one classification for Chinese Mandarin and test it with a telephone conversation corpus, using a weak supervision method to build a model for automatic labeling.

In chapter 2 we do the literature review about dialogue act and NSU, which is necessary for understanding the role and importance of NSU. Also, the literature of dialogue act combined with machine learning methods are enlightening for our work of NSU.

In chapter 3 we present the weak supervision method used in this work; Snorkel's key is to write practical Labeling functions using linguistic expertise and apply its built-in algorithm to optimize the result. We do the descriptive statistics of our data, using some graphs to present their distributions of several aspects, especially for the timing cues. We also see some characteristics of our sample data to have some initial ideas about the difference between the NSU labeled data and the sentential utterances as well as the disfluency data.

In chapter 4 we see a qualitative analysis of the NSUs in the sample data, from which we see the characteristics of each class and compare it with the definition. Some classes like the modifiers do not conform precisely to the description. We also reflect upon the classification of Wong to see if the added types are pertinent.

In chapter 5, we present features and cues used for this experiment; those are revealed in the Labelling functions. Then we describe the operation of the NSU's detection task in the corpus, based on some common attributes of the NSU, we managed to distinguish it from the sentential utterance and others. Besides the NSU, there are sentential utterances, disfluency and others.

In chapter 6, we illustrate the classification task, as the performance with all the labels is hard to achieve satisfaction; we also present results with the major categories.

Future development Regarding the imbalance of the labels, we can attend more to the less present data and find more labels. We can find more instances of the minor categories and adjust the proportion inside the set with admissible labels.

For the categories based heavily on the semantic knowledge, we can try to comply with the framework and classifications matched, such as the work of (Schlangen, 2003).

To enrich the cues and to improve the performance of our model, we can introduce some audio-related features in our work, such as the prosodic cues used in (Stolcke et al., 2000).

We did not use third-party models in this study, but maybe we can test them in future works with appropriate corpus, such as sentimental analysis and semantic similarity.

Bibliography

Abeillé, A., Delaveau, A., and Godard, D. (2007). La grande grammaire du français : principes de construction. *Revue roumaine de linguistique*, LII:403–419.

Asher, N. and Lascarides, A. (2003). *Logics of Conversation*. Cambridge University Press, http://www.cambridge.org/.

Austin, J. L. (1962). *How to do things with words*. William James Lectures. Oxford University Press.

Barton, E. (1991). Nonsentential Constituents and Theories of Phrase Structure. In Leffel, K. and Bouchard, D., editors, *Views on Phrase Structure*, pages 193–214. Springer Netherlands, Dordrecht.

Blanche-Benveniste, C. (1997). *Approches de la langue parlée en français*. Collection L'essentiel français. Ophrys, Gap Paris. graph. 21 cm. Bibliogr. p. 149-151. Index.

Blanche-Benveniste, C. (2010). *Approches de la langue parlée [en français]*. L'essentiel français. Ophrys, Paris, nouv. éd edition. OCLC: 845869099.

Bunt, H. (1994). Context and dialogue control. *THINK Quarterly*, 3.

Bunt, H. (1995). Dialogue control functions and interaction design. In *Dialogue in Instruction*, pages 197–214. Springer Verlag.

Bunt, H. (2005). A framework for dialogue act specification.

Bunt, H. (2006). Dimensions in dialogue act annotation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

Bunt, H. (2017). Computational Pragmatics.

Bunt, H., Alexandersson, J., Carletta, J., Choe, J.-W., Fang, A. C., Hasida, K., Lee, K., Petukhova, V., Popescu-Belis, A., Romary, L., Soria, C., and Traum, D. (2010). Towards an ISO standard for dialogue act annotation. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Chollet, F. et al. (2015). Keras. https://keras.io.

Cilibrasi, R. L. and Vitanyi, P. M. B. (2007). The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383.

Core, M. G. and Allen, J. (1997). Coding dialogs with the damsl annotation scheme. In *AAAI fall symposium on communicative action in humans and machines*, volume 56, pages 28–35. Boston, MA.

Dragone, P. (2015). Non-sentential utterances in dialogue: Experiments in classification and interpretation. *CoRR*, abs/1511.06995.

Fang, A., Li, Y., Cao, J., and Bunt, H. (2019). *Chinese multimodal resources for dialogue act analysis*, pages 256–275. Routledge, 1st edition.

Fernández, R. and Ginzburg, J. (2002). Non-sentential utterances in dialogue: A corpus-based study. In *Proceedings of the Third SIGdial Workshop on Discourse and Dialogue*, pages 15–26, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Fernández, R., Ginzburg, J., and Lappin, S. (2007). Classifying Non-Sentential Utterances in Dialogue: A Machine Learning Approach. *Computational Linguistics*, 33(3):397–427.

Fišel, M. (2007). Machine learning techniques in dialogue act recognition. *Eesti Rakenduslingvistika Ühingu aastaraamat Estonian Papers in Applied Linguistics*, 0(3):117–134.

Garcia-Marchena, O. (2015). Phrases averbales et fragments en espagnol oral. Rome. University Roma Sapienza.

Ginzburg, J. (2012). *The Interactive Stance: Meaning for Conversation*. Oxford University Press UK.

Harvey, K. (2011). The Language of Outsourced Call Centres: A Corpus-Based Study of Cross-Cultural Interaction by Eric Friginal. *Journal of Sociolinguistics*, 15(4):528–532.

Heeman, P. and Allen, J. (1994). Detecting and correcting speech repairs. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 295–302, Las Cruces, New Mexico, USA. Association for Computational Linguistics.

Heldner, M. and Edlund, J. (2010). Pauses, gaps and overlaps in conversations. *Journal of Phonetics*, 38(4):555 – 568.

Huang, S. (1999). The emergence of a grammatical category definite article in spoken Chinese. *Journal of Pragmatics*, 31(1):77–94.

Jurafsky, D., Bates, R., Coccaro, N., Martin, R., Meteer, M., Ries, K., Shriberg, E., Stolcke, A., Taylor, P., and Van Ess-Dykema, C. (1997). Automatic detection of discourse structure for speech recognition and understanding. In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pages 88–95.

Jurafsky, D., Shriberg, E., Fox, B., and Curl, T. (1998). Lexical, prosodic, and syntactic cues for dialog acts. In *Discourse Relations and Discourse Markers*.

Levinson, S. C. (1983). *Pragmatics*. Cambridge Textbooks in Linguistics. Cambridge University Press.

Liu, B. (2009). Chinese Discourse Markers in Oral Speech of Mainland Mandarin Speakers. In *Proceedings of the 21st North American Conference on Chinese Linguistics (NACCL-21). Volume 2*, pages 358–374, Smithfield, Rhode Island: Bryant University.

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

Muller, P. and Prevot, L. (2009). Grounding information in route explanation dialogues. In *Spatial Language and Dialogue*. Oxford University Press.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Édouard Duchesnay (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830.

Pérez, F. and Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29.

Prevot, L. (2004). *Semantic and Pragmatic structures for modelling coherence in task-oriented dialogues*. Theses, Université Paul Sabatier - Toulouse III.

Pénault, A. (2018). *Les énoncés non-phrastiques en français oral : une analyse intégrant les niveaux syntaxique, sémantique et interactionnel*. PhD thesis. Thèse de doctorat dirigée par Prévot, Laurent Sciences du langage Aix-Marseille 2018.

Ratner, A., Bach, S. H., Ehrenberg, H. R., Fries, J. A., Wu, S., and Ré, C. (2017a). Snorkel: Rapid training data creation with weak supervision. *CoRR*, abs/1711.10160.

Ratner, A., Hancock, B., Dunnmon, J., Sala, F., Pandey, S., and Ré, C. (2018). Training Complex Models with Multi-Task Weak Supervision. *arXiv:1810.02840 [cs, stat]*. arXiv: 1810.02840.

Ratner, A., Sa, C. D., Wu, S., Selsam, D., and Ré, C. (2016). Data programming: Creating large training sets, quickly. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3574–3582, Red Hook, NY, USA. Curran Associates Inc.

Ratner, A. J., Bach, S. H., Ehrenberg, H. R., and Ré, C. (2017b). Snorkel: Fast Training Set Generation for Information Extraction. In *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*, pages 1683–1686, Chicago, Illinois, USA. ACM Press.

Sacks, H., Schegloff, E. A., and Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language*, pages 696–735.

Sasaki, Y. (2007). The truth of the F-measure. page 5.

Schlangen, D. (2003). A Coherence-Based Approach to the Interpretation of Non-Sentential Utterances in Dialogue.

Schlangen, D. (2005). Towards finding and fixing Fragments—Using ML to identify non-sentential utterances and their antecedents in multi-party dialogue. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 247–254, Ann Arbor, Michigan. Association for Computational Linguistics.

Schlangen, D. and Lascarides, A. (2003). The interpretation of non-sentential utterances in dialogue. page 10.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

Shriberg, E. (1996). Disfluencies in switchboard. In *Proceedings of the International Conference on Spoken Language Processing*, pages 11–14, Philadelphia, PA.

Stalnaker, R. (2002). Common Ground. *Linguistics and Philosophy*, 25(5-6):701–721.

Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., Taylor, P., Martin, R., Van Ess-Dykema, C., and Meteer, M. (2000). Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3):339–374.

Tao, H. and Thompson, S. A. (1991). English backchannels in mandarin conversations: A case study of superstratum pragmatic 'interference'. *Journal of Pragmatics*, 16(3):209 – 223.

Tseng, S.-C. (1999). *Grammar, Prosody and Speech Disfluencies in Spoken Dialogues*. PhD thesis, University of Bielefeld.

Tseng, S.-C. (2003). Repairs and repetitions in spontaneous mandarin. In *ISCA Tutorial and Research Workshop on Disfluency in Spontaneous Speech*.

Wong, K.-C. (2018). *Classifying Conversations*. PhD thesis, Université Paris Diderot - Paris 7.

Wong, K.-C. and Ginzburg, J. (2013). Investigating non-sentential utterances in a spoken chinese corpus.

Wong, K.-C. and Ginzburg, J. (2018). Conversational types: a topological perspective. In *Proceedings of the 22nd Workshop on the Semantics and Pragmatics of Dialogue - Full Papers*, Aix-en-Provence, France. SEMDIAL.

Xia, F. (2000). The part-of-speech tagging guidelines for the penn chinese treebank (3.0. Technical report, Linguistic Data Consortium.

Xudong, D. (2008). The use of listener responses in Mandarin Chinese and Australian English conversations. *Pragmatics*, 18(2):303–328.

Yngve, V. H. (1970). On getting a word in edgewise. In *CLS-70*, pages 567–577. University of Chicago.

Yu-min, L. (2010). On the establishment of the audio database of chinese language resources.

Zhang, Y. and Clark, S. (2011). Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Zhou, J., Narentuya, and Shi, J. (2015). A semantic study of modal words in mandarin chinese. In Lu, Q. and Gao, H. H., editors, *Chinese Lexical Semantics*, pages 160–172, Cham. Springer International Publishing.

Appendix A

Appendix

In this notebook, we want to do Exploratory Data Analysis for our corpus CallHome

The data we used in this study is from LDC's CALLHOME Mandarin Chinese collection (https://catalog.ldc.upenn.edu/LDC96T16). This is a telephonic conversation corpus, with audio files and transcriptions. The language was in Mandarin even though the participants are from different provinces of China.

1. Set the environment, import the relative modules and the data

In [1]:

```
import sys
print(sys.version)
#import spacy
#print(spacy.__version__)
```

```
3.7.3 (default, Dec 13 2019, 19:58:14)
[Clang 11.0.0 (clang-1100.0.33.17)]
```

In [2]:

```
import re
import string
import time
import nltk

import numpy as np
import pandas as pd


import seaborn as sns

import cycler

import matplotlib.pyplot as plt
```

The Preprocessing of the data is not included in this notebook.

The original data includes the start time and end time of every segment, the speaker(A, B or B1, etc.), the textual content of the utterance. In the text transcription, there are also examples of annotation as enrichment of information. These original information are divided into four columns.

The original information is separated into four columns: Start time, End time, Speaker, and Text transcription. Based on these, we added other columns :

- Conversation code: the original code of the file
- Duration : how long the utterance lasts
- Same Speaker: if the utterance is produced by the speaker of the previous utterance
- Latency : a gap between two turns, the Start time minus the previous End time (we only consider the positive value cases)
- Overlap : the duration when more than one person speaks (we only consider the positive value cases)

- Word count: How many units are there in the utterance (depending on the segmentation method, one unit may not necessary correspond to one Chinese character, and the punctuation can be included as well)
- POS tags: the POS tags used in this study is attributed by the tool Zpar (https://pypi.org/project/python-zpar/)

After preprocessing, we have the data in the right form, we import the data from a local directory.

In [3]:

```
df_callhome_all = pd.read_csv('Data/calltags_0519.csv')
```

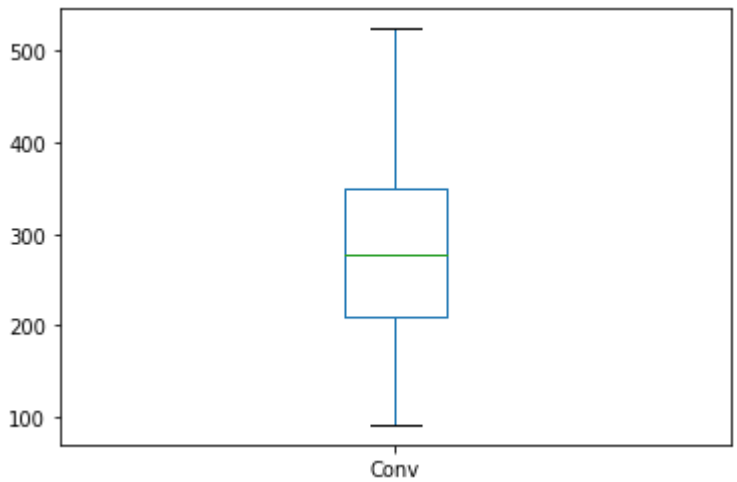
We've done some descriptive statistics of the whole data and look at some random individual files. We have 33485 lines of utterances in total in combining 120 files. We can see the distribution of utterances count among the 120 files, most of them are between 200-400 turns.

In [4]:

```
df_callhome_all['Conv'].value_counts().plot.box()
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x122c60e10>
```



Another way to visualize the distribution using historgram

In [5]:

```
df_ipu_number = df_callhome_all.groupby('Conv')['Conv'].size()
```

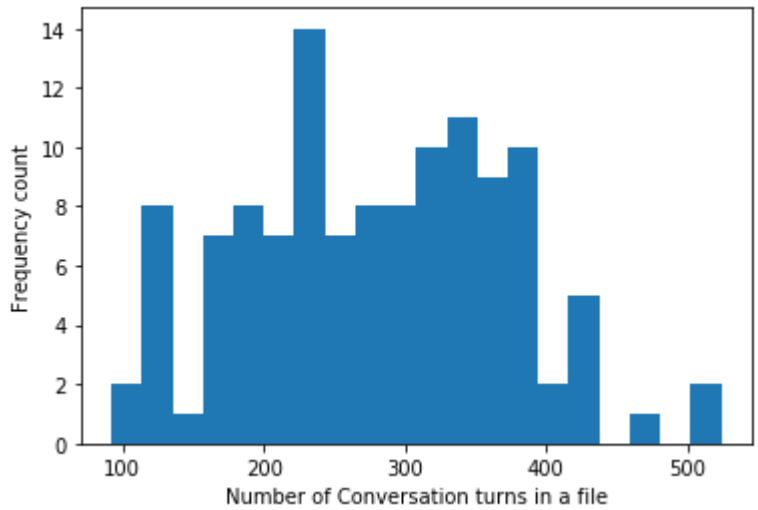
In [6]:

```
ax = df_ipu_number.plot(kind='hist',bins=20)

ax.set_ylabel('Frequency count')
ax.set_xlabel('Number of Conversation turns in a file')
```

Out[6]:

```
Text(0.5, 0, 'Number of Conversation turns in a file')
```



In [7]:

```
df_callhome_all.describe()
```

Out[7]:

	Unnamed: 0	ID	Conv	Duration	End	Latency	
count	33432.000000	33432.000000	33432.000000	33432.000000	33432.000000	19777.000000	243
mean	16732.459141	16733.459141	909.230468	1.915543	415.813732	0.606402	
std	9664.742222	9664.742222	351.661536	1.900217	173.846619	0.856297	
min	0.000000	1.000000	3.000000	0.090000	15.900000	0.000000	
25%	8366.750000	8367.750000	748.000000	0.580000	267.917500	0.150000	
50%	16726.500000	16727.500000	821.000000	1.340000	410.840000	0.390000	
75%	25089.250000	25090.250000	977.000000	2.590000	563.370000	0.770000	
max	33484.000000	33485.000000	1737.000000	36.340000	851.700000	25.150000	

2. Visualisation of numerical columns

The table above has already all the useful values, next we use some visualisations to have a more direct idea.

We see the distribution of the four numerical columns : Latency, Overlap, Word_count, Duration.

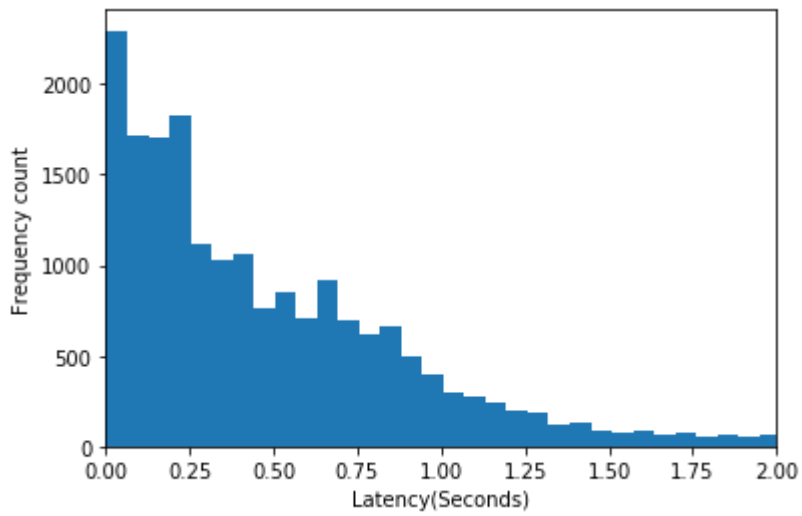
In [8]:

```
axlatency = df_callhome_all['Latency'].plot(kind="hist",bins=400,xlim=(0,2))

axlatency.set_ylabel('Frequency count')
axlatency.set_xlabel('Latency(Seconds)')
```

Out[8]:

```
Text(0.5, 0, 'Latency(Seconds)')
```



If we exclude the value of 0 in Latency, then :

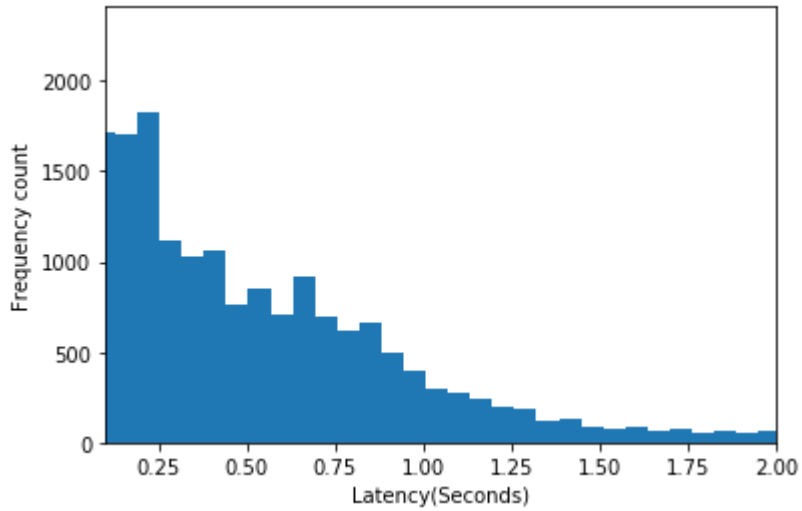
In [9]:

```
axlatency = df_callhome_all['Latency'].plot(kind="hist",bins=400,xlim=(0.1,2))

axlatency.set_ylabel('Frequency count')
axlatency.set_xlabel('Latency(Seconds)')
```

Out[9]:

```
Text(0.5, 0, 'Latency(Seconds)')
```



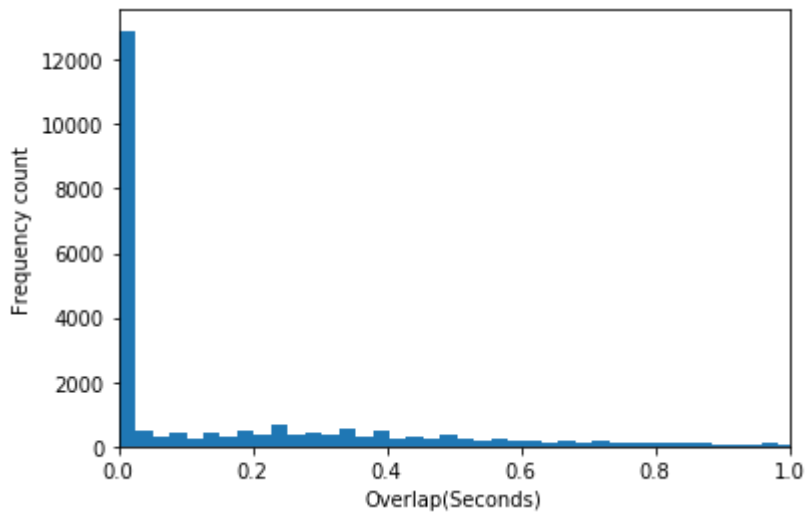
In [10]:

```
axoverlap = df_callhome_all['Overlap'].plot(kind='hist',bins=200,xlim=(0,1))

axoverlap.set_ylabel('Frequency count')
axoverlap.set_xlabel('Overlap(Seconds)')
```

Out[10]:

```
Text(0.5, 0, 'Overlap(Seconds)')
```



If we exclude the value of 0 in Overlap, then :

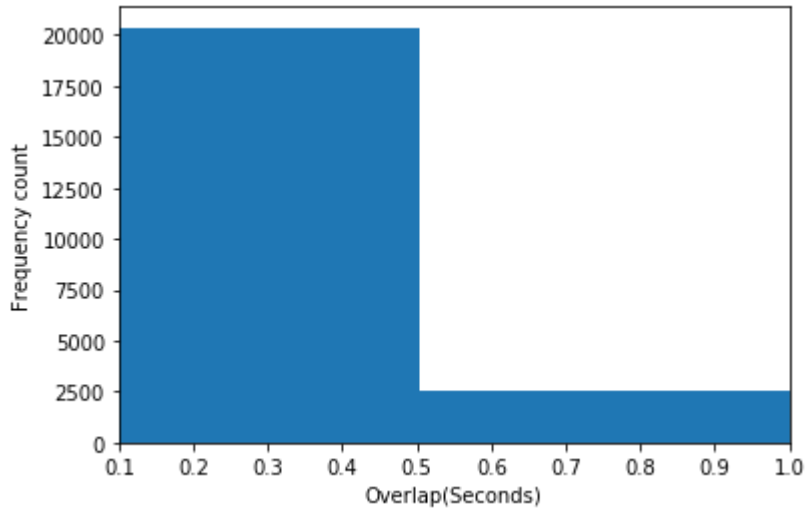
In [11]:

```
axoverlap = df_callhome_all['Overlap'].plot(kind='hist',bins=10,xlim=(0.1,1))

axoverlap.set_ylabel('Frequency count')
axoverlap.set_xlabel('Overlap(Seconds)')
```

Out[11]:

```
Text(0.5, 0, 'Overlap(Seconds)')
```

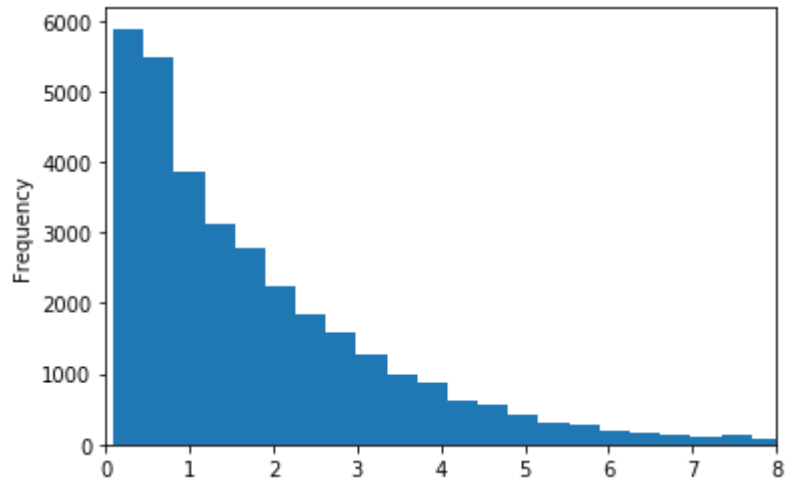


In [12]:

```
df_callhome_all['Duration'].plot(kind="hist",bins=100,xlim=(0,8))
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x123e11ef0>
```



We can also see the distribution of certain column(s) in a certain file

In [13]:

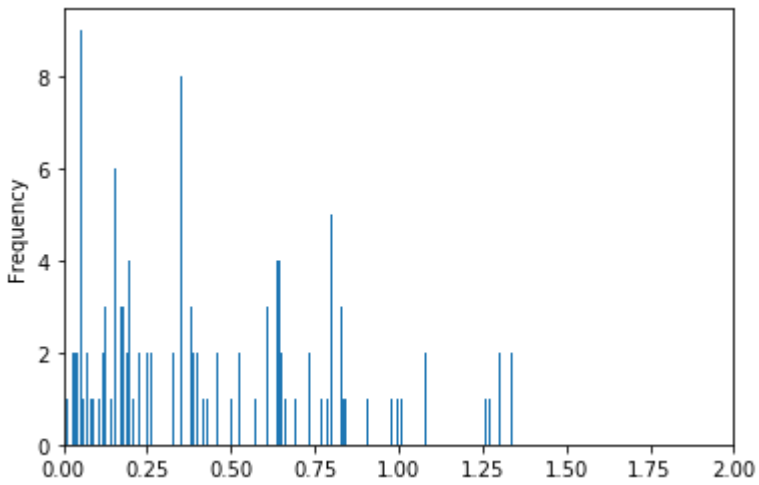
```
call0003 = df_callhome_all.loc[df_callhome_all['Conv'] == 3]
```

In [14]:

```
call0003['Latency'].plot(kind='hist',bins=400,xlim=(0, 2))
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x123fbd588>
```



In [15]:

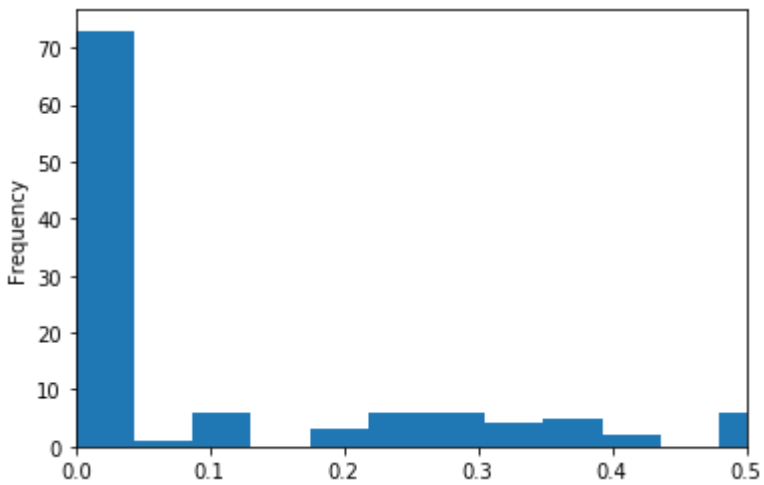
```
call0022 = df_callhome_all.loc[df_callhome_all['Conv'] == 22]
```

In [16]:

```
call0022['Overlap'].plot(kind='hist',bins=50,xlim=(0,0.5))
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1246cc908>
```



We didn't forget about the column Word_count:

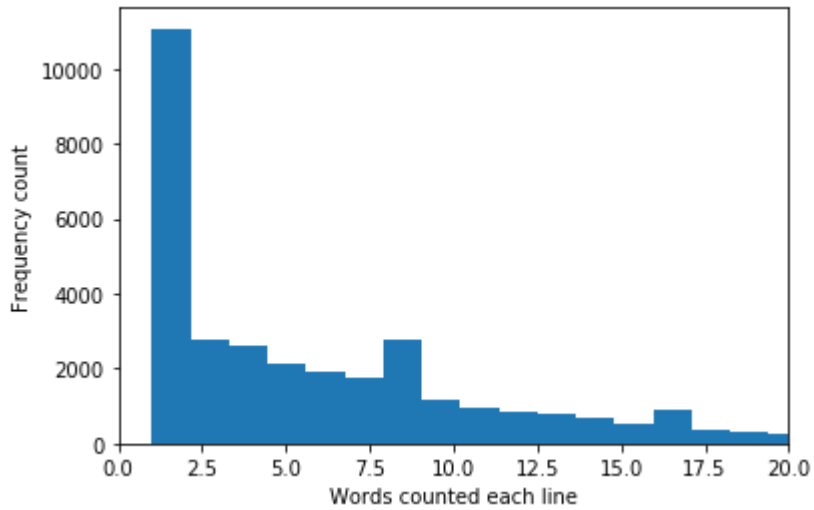
In [17]:

```
axword = df_callhome_all['Word_count'].plot(kind="hist",bins=100,xlim=(0,20))

axword.set_ylabel('Frequency count')
axword.set_xlabel('Words counted each line')
```

Out[17]:

```
Text(0.5, 0, 'Words counted each line')
```



This graph shows the majority of the value "1" in "Word_count", if we look at this subset, we may find some of them are the same: "啊 ? "(Ah ?), "呃 ,"(Uh ,), "嗯 ,"(Um ,)", "哦 ,"(oh ,) We need to point out here that the values are counted base on the segmentation method, so the punctuation could be counted as one, and some pattern containing more than one Chinese character could be counted as one, for example : "对呀 ."(2 characters),"真是的 ."(3 characters),"对不对 ?"(3 characters)

Another small peak at this graph is the value 8, but in the text columns we can't really have get some frequent patterns.

Below is the subset of the data when the "Word_count" = 1.

In [18]:

```
callone = df_callhome_all[df_callhome_all['Word_count']== 1]
```

In [19]:

```
callone['Text'].value_counts()
```

Out[19]:

```
嗯 .                    951
{laugh}                633
哦 .                    523
呃 ,                    474
嗯 ,                    457
啊 .                    398
呃 .                    324
啊 ?                    311
哎 ,                    310
哦 ,                    299
哎 .                    242
对 .                    225
嗷 ,                    189
对 ,                    165
(())                   140
是吗 ?                    88
啊 ,                     81
行 .                     52
啊哈 .                    42
是吧 ?                    42
{cough}                 40
喔 .                     39
喂 ,                     38
[noise]                 36
呀 .                     34
嗯 ?                     34
好 .                     32
对呀 ,                    30
唉 .                     29
对呀 .                    29
                       ...
((呵))                    1
((不行)) ,                 1
耶 ,                      1
地址 .                     1
办公室 ,                    1
每次 ,                     1
%嗯% ,                     1
他-                       1
没 .                      1
确实 .                     1
不要紧 ,                    1
((你)) .                   1
当然 ,                     1
<English_Hello,>          1
啊哈 ,                     1
那-                       1
转-                       1
<English_Mil->            1
女朋友 ?                    1
&琉& .                     1
忙 .                      1
((比-))                    1
<English_L_O_O_S>         1
几号 ?                     1
```

```
((行)) .                  1
<English_S_T>             1
<English_So_long.>        1
((咦)) .                  1
能够 ,                    1
说 ,                      1
Name: Text, Length: 622, dtype: int64
```

Take a look at the small peak when column Word_count = 8

In [20]:

```
df_callhome_all[df_callhome_all['Word_count']== 8]
```

Out[20]:

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	Start	
37	37	38	882	1.67	280.27	0.20	0.00	False	A	278.60	嗯 那个
40	40	41	882	1.59	285.94	0.06	0.00	False	B	284.35	啊 , 下
95	95	96	882	2.42	407.11	0.40	0.00	False	A	404.69	我 <Engl

Plot the mean of numerical columns for each file (120 in total)

When we look for all values in the whole dataset, the means of Latency and Overlap are 0.60 and 0.24, so generally, when there is an overlap, it is rather short, and the latency between two turns can be more evident in this corpus. This may be due to the relation between the speakers(family member) and the nature of the conversation(telephonic conversation). We plot the means of each file for the Latency and Overlap columns, and we can see they are a bit close to Gaussian distribution.

In [21]:

```
### Plot means
```

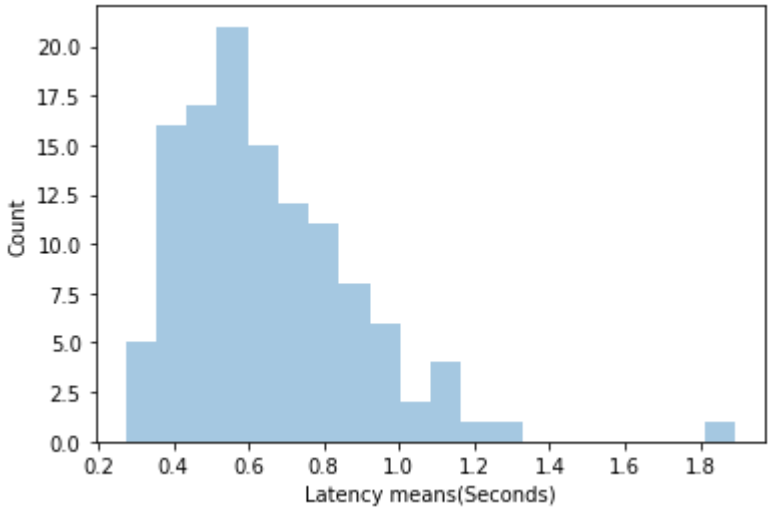
In [22]:

```
latency_means = df_callhome_all.groupby('Conv')['Latency'].mean()
```

In [23]:

```
axmla = sns.distplot(latency_means,bins=20,kde=False)

axmla.set(xlabel='Latency means(Seconds)', ylabel='Count')
plt.show()
```



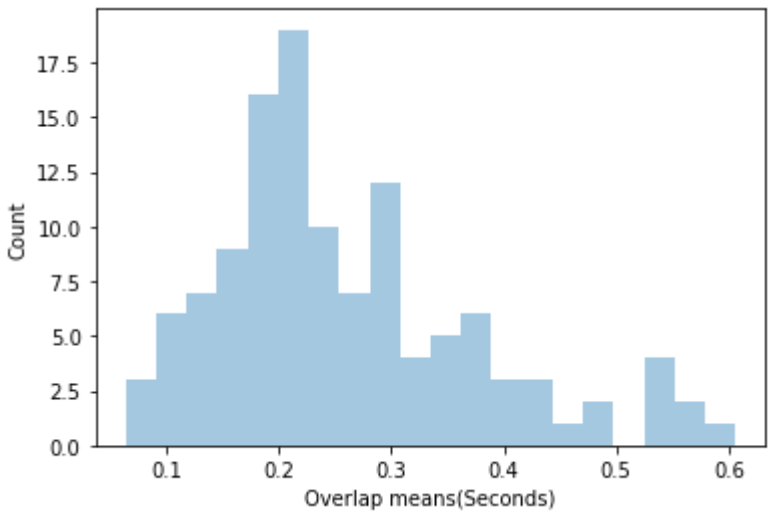
In [24]:

```
overlap_means = df_callhome_all.groupby('Conv')['Overlap'].mean()
```

In [25]:

```
axmov = sns.distplot(overlap_means,bins=20,kde=False)

axmov.set(xlabel='Overlap means(Seconds)', ylabel='Count')
plt.show()
```



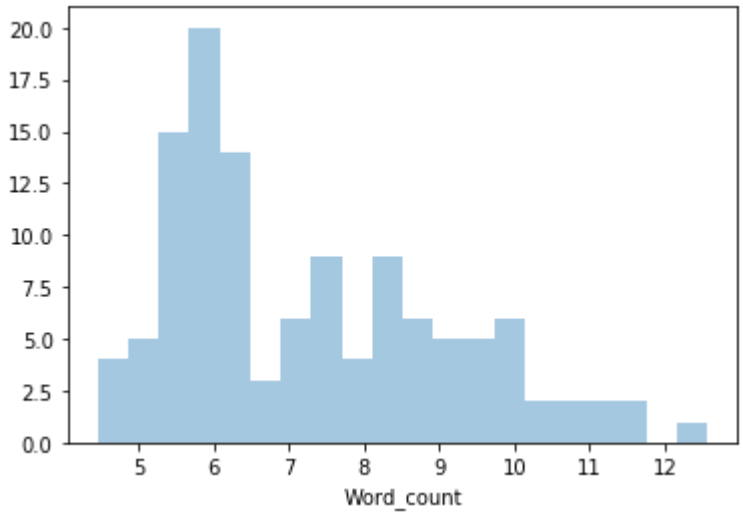
In [26]:

```
countwords_means = df_callhome_all.groupby('Conv')['Word_count'].mean()
```

In [27]:

```
axcw = sns.distplot(countwords_means,bins=20,kde=False)

axmov.set(xlabel='Means of words in each utterances', ylabel='Count')
plt.show()
```



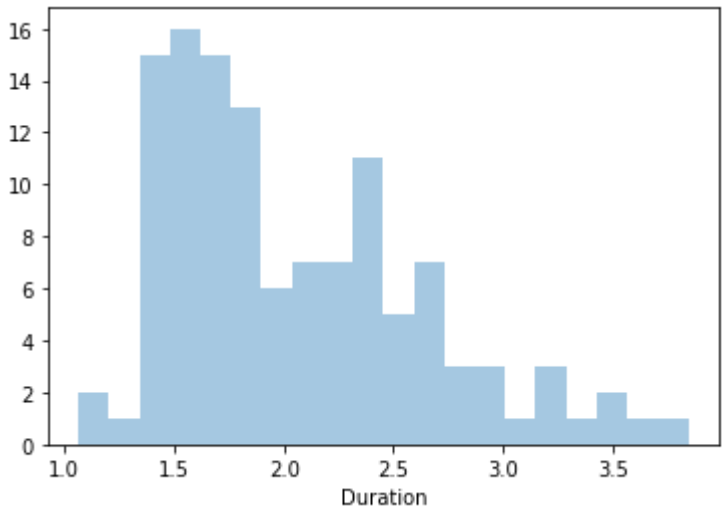
In [28]:

```
duration_means = df_callhome_all.groupby('Conv')['Duration'].mean()
```

In [29]:

```
axdura = sns.distplot(duration_means,bins=20,kde=False)

axmov.set(xlabel='Duration(Seconds)', ylabel='Count')
plt.show()
```



We can observe the main speaker in a certain conversation by observing some parameters

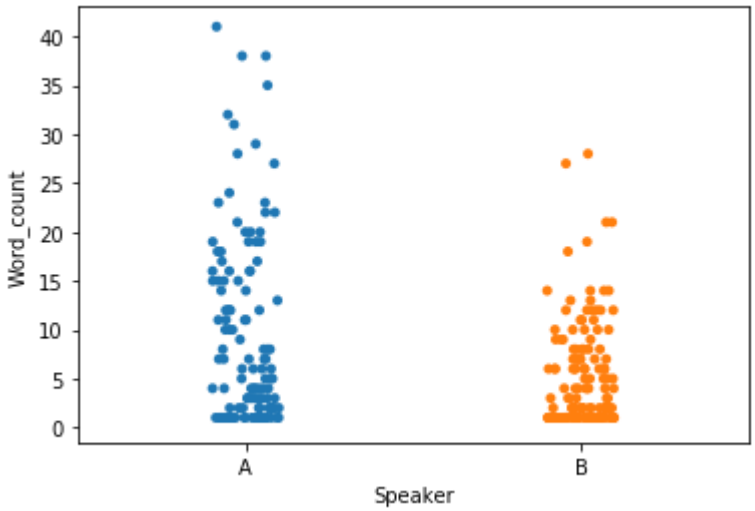
In the graph below, we look at the Word_count column in a file coded 1376. We see that speaker A produces longer phrases, A may be the main speaker because he produces long sentences, and B generally provides shorter phrases. Still, we can't be sure, and this is only one dimension so that we may need the data in other aspects too.

In [30]:

```
call1376 = df_callhome_all.loc[df_callhome_all['Conv'] == 1376]
```

In [31]:

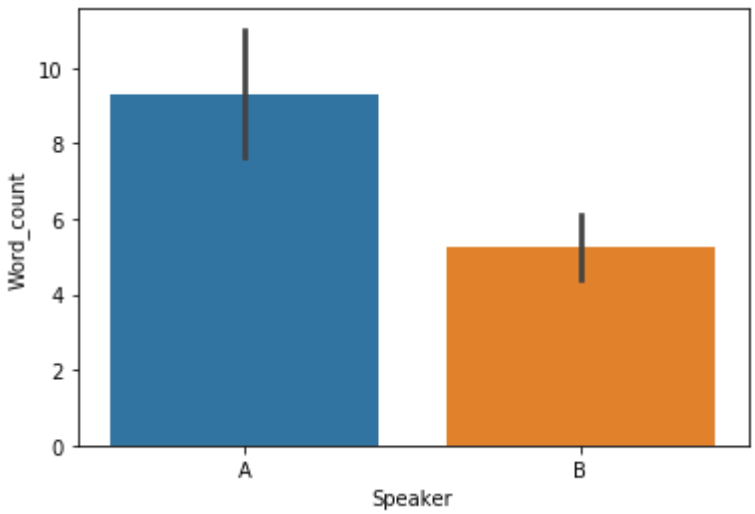
```
sns.stripplot(x='Speaker', y='Word_count', data=call1376)
plt.ylabel('Word_count')
plt.show()
```



Just another alternative of visualisation, it's more intuitive to see the discrepancy between the two speakers, but the density side is hidden.

In [32]:

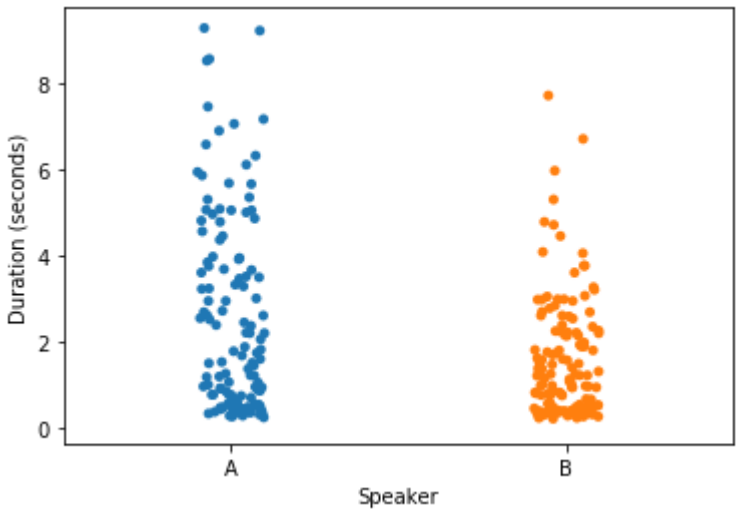
```
sns.barplot(x='Speaker', y='Word_count', data=call1376)
plt.ylabel('Word_count')
plt.show()
```



The duration doesn't offer much new information compared with the "Word_count" graph, it's comprehensible, because they are very similar.

In [33]:

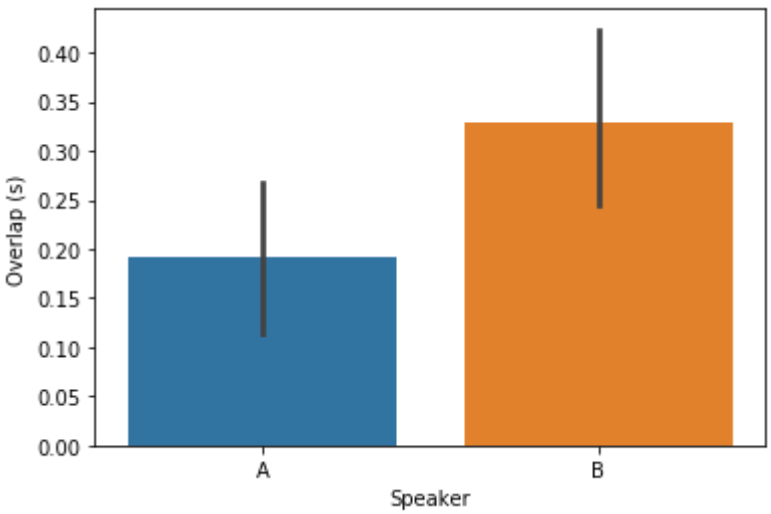
```
sns.stripplot(x='Speaker', y='Duration', data=call1376)
plt.ylabel('Duration (seconds)')
plt.show()
```



We see that B has more Overlaps even if the actual differences in number is small. So it's adding to our hypothesis that A is the main speaker, and B provided more feedbacks when the dominant speaker has the floor.

In [34]:

```
sns.barplot(x="Speaker", y="Overlap", data=call1376)
plt.ylabel('Overlap (s)')
plt.show()
```



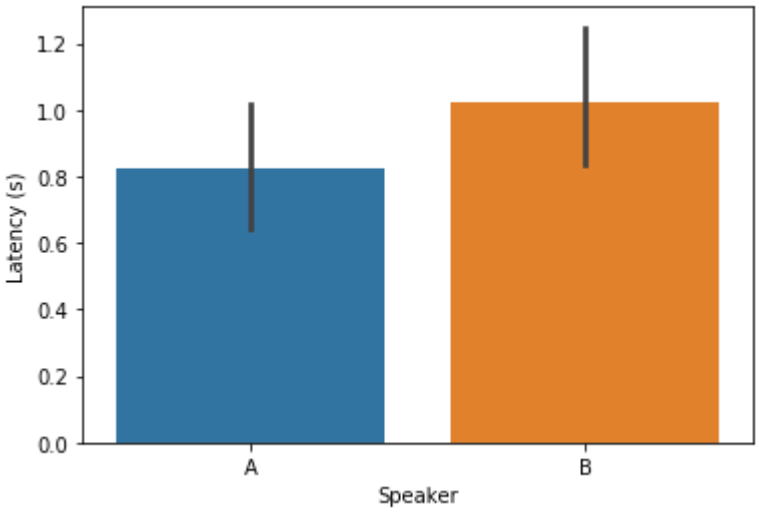
The last aspect is the Latency, B also has a bigger value of latency compared to A, that B waits longer time if the previous speaker is A. Along with the observation for this file in other aspects, we can be rather sure that A is the main speaker.

We may have the reasonable summary that, if one is the main speaker in a conversation, it has bigger value of Word_count and Duration, it has smaller value in terms of Latency and Overlap.

This hypothesis can be useful and we can go further to set some numerical scope for this hypothesis.

In [35]:

```
sns.barplot(x="Speaker", y="Latency", data=call1376)
plt.ylabel('Latency (s)')
plt.show()
```



We can also see a group of graphs togerther using pairplot.

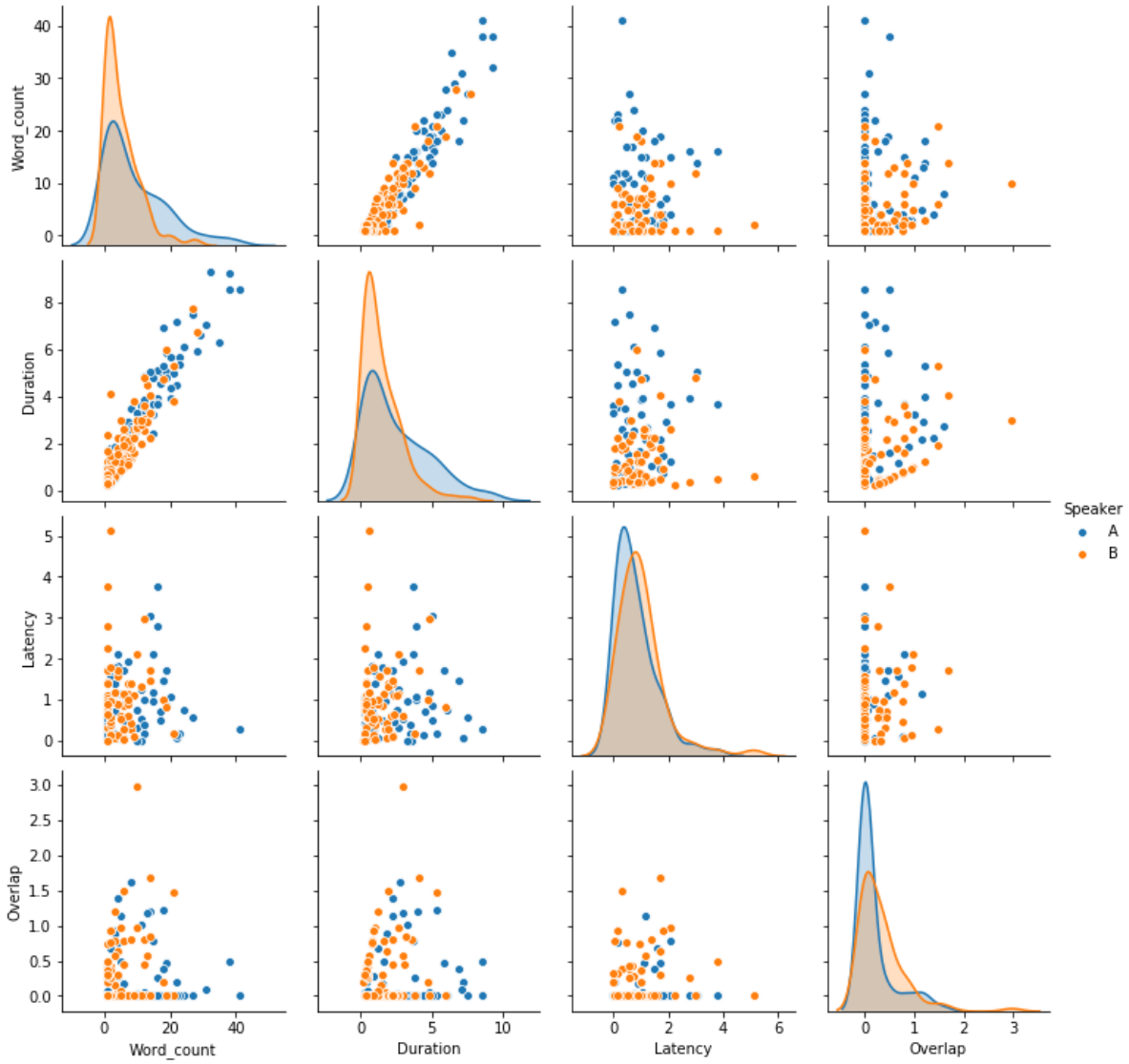
We can tell the dominant speaker and some characteristics from this graph below, seen from the x axis, and look at the line graphs, like the Word_count column, we see that the speaker A produce longer utterances than B, what's more, for the duration it's also similar so generally A has the floor for the most of time.

Then for the Latency, compared with B, A waits slightly less time than B when there is a blank between two turns of speech, on contrary B has a high proportion of overlap but the overlap lasts for a short time, we can infer that it's mostly backchannel.

From this we can be rather sure than the A is the dominant speaker. But this kind of analysis may not apply to all the conversations because there are individual differences. Also, if the relation between the two speakers are different, like superior and subordinate, teacher and student, the proportion can be very different.

In [36]:

```
graph_1376 = sns.pairplot(call1376[['Speaker','Word_count','Duration','Latency','Ove
#labels = graph_1376._legend_data.keys()
plt.show()
```



When there are more than two speakers in a conversation, the situation may be a little more complicated. From the scatter plots, we can see that A and B are more present than B1 and B2.

From the line plots, at first sight, we see that for the Word_count column, B produces longer utterances than the rest of the speakers, so we can assume B is the main speaker in this conversation, and B2 has the most extended duration among the four speakers, the next one is B.

B has a shorter Overlap than A; possibly, A produces more feedback to B. In terms of Latency A and B are similar.

In [37]:

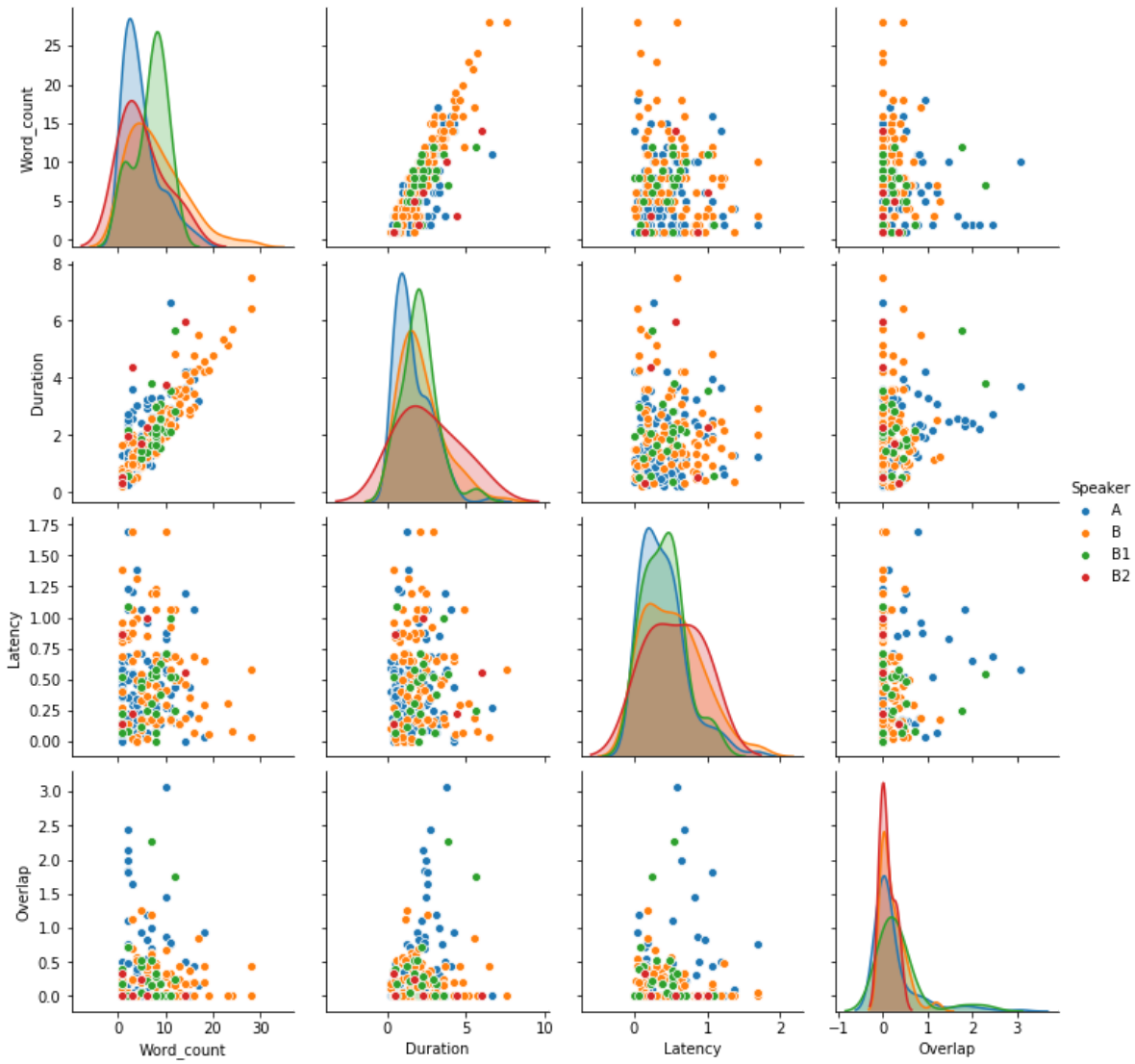
```
call0861 = df_callhome_all.loc[df_callhome_all['Conv'] == 861]
```

In [38]:

```
sns.pairplot(call0861[['Speaker','Word_count','Duration','Latency','Overlap']], hue=
#labels = graph_1376._legend_data.keys()
#plt.show()
```

Out[38]:

```
<seaborn.axisgrid.PairGrid at 0x1250cac50>
```



3. A pilot study with 5% of the corpus

We also select around 5% of the whole data(the first 1500 lines of the entire information) to tag the data manually to have an idea about the difference between data with NSU tags and the whole data. We expect that the numerical columns may behave differently. Because the NSUs are generally short utterances so the duration might be brief, and the counted words may be less than the mean of all the data. Same for the Latency and overlap, so we want to know the exact difference. Below is the results.

In [39]:

```
call1500 = pd.read_csv("data/label_0522.csv")
```

Extract the data with NSU labels; this time, we assign the non-NSU to 0, so it's all the non zero labels. It should be noted that a non-NSU type"disfluency" are labeled 30, we give it a label because they are generally short utterances, if we only look at the duration and Word_count, they are easily confounded with the NSU. Having this class is useful for our task of detection of NSU, but now we just look at the NSU data.

In [40]:

```
call_tags = call1500[call1500["Label"]!= 0 ]
```

For comparison, we also sort out the sentential utterance data(SU) and the disfluency data in the sample corpus.

In [41]:

```
call_su = call1500[call1500["Label"] == 0 ]
```

In [43]:

```
call_disfluency = call1500[call1500["Label"] == 30 ]
```

In [45]:

```
call_nsu = call_tags[call_tags["Label"]!= 30]
```

In [46]:

```
len(call_nsu)
```

Out[46]:

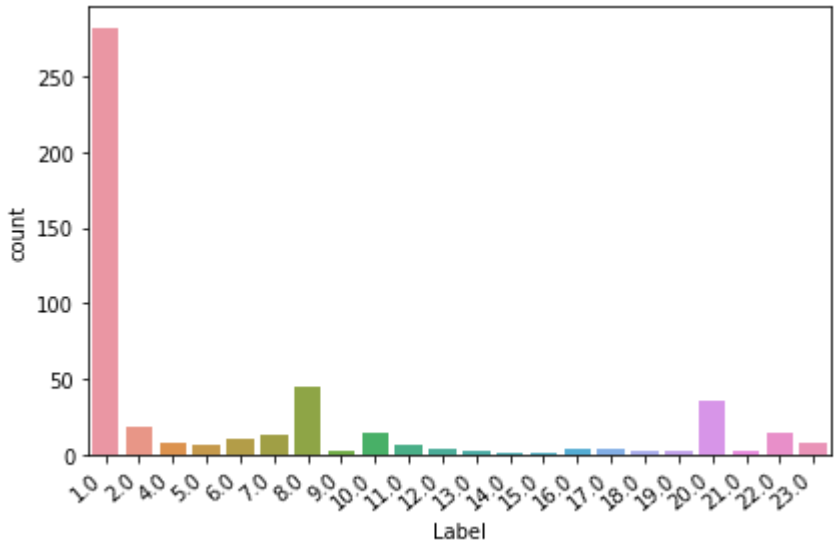
```
490
```

Then we look at the distribution of labels, the label 1 has overwhelming majority , we can also leave the label one to see other important classes more clearly. But the conclusion is there are 4 major classes : label 1, 2, 8, 20.

In [47]:

```
axnsu = sns.countplot(x='Label',data=call_nsu)

axnsu.set_xticklabels(axnsu.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



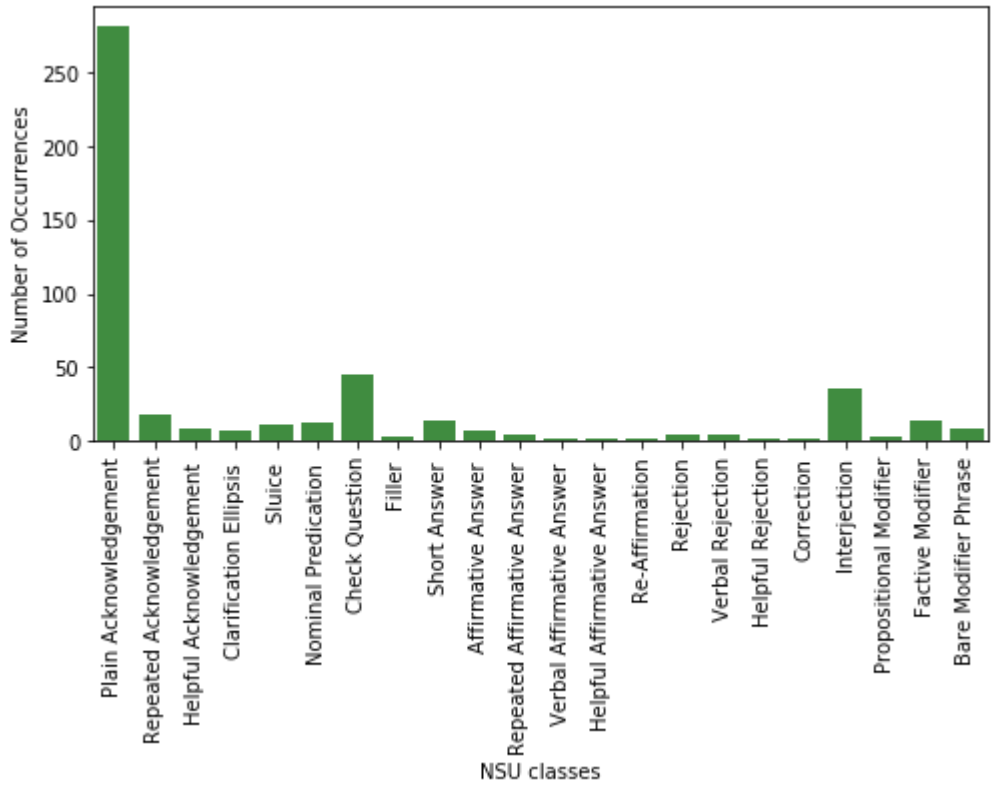
Each number corresponds to an NSU tag, as the numbers may be confusing, so let's add the textual names of the labels.

In [48]:

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

color = ['r', 'g', 'b']
label_text = pd.read_csv("data/label_text.csv")
is_dup = call_nsu['Label'].value_counts()

plt.figure(figsize=(8,4))
ax = sns.barplot(is_dup.index, is_dup.values, alpha=0.8, color=color[1])
ax.set_xlabel('NSU classes')
ax.set_ylabel('Number of Occurrences')
ax.set_xticklabels(label_text['labels'], rotation='vertical', fontsize=10)
plt.show()
```



Compare the distributions of the numerical columns of the sample nsu data and the original data

The form of the box plots includes five values, the upper line of the box is 75%, the bottom edge of the box means the 25% value, the inside track means the average number. The bottom of the whisker refers to the minimum amount, and the top of the whisker means the maximum value.

For the NSU labeled data:

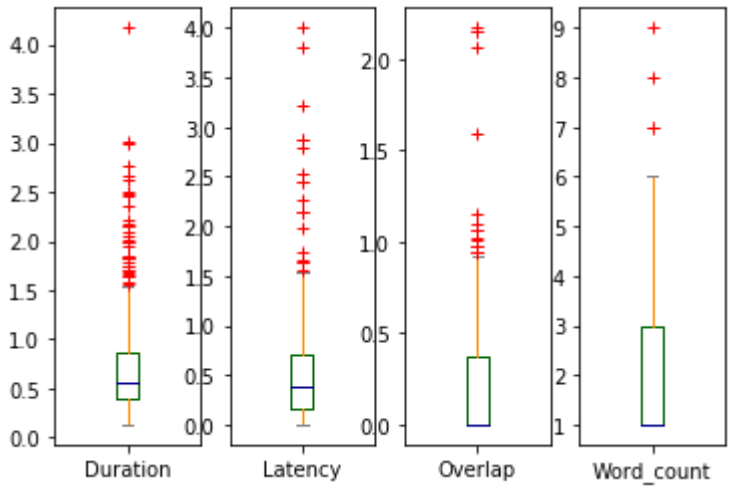
- The Duration box is between 0.4 and 1, almost all below 4.
- The Latency box is around 0.25 to 0.7, and almost all of them are less than 4.
- The overlap box is from 0 to 0.4; most of them are less than 2.
- The Word_count column's box is from 1 to 3 and generally less than 6.

In [49]:

```
# Make a list of the column names to be plotted: cols
cols = ['Duration','Latency','Overlap','Word_count']
color = dict(boxes='DarkGreen', whiskers='DarkOrange',
             medians='DarkBlue', caps='Gray')


# Generate the box plots
call_nsu[cols].plot(kind = 'box',subplots=True, color = color,sym='r+')


# Display the plot
plt.show()
```



For the aggregate data, we can see that almost all the columns have a long tail representing a lot of outliers if we look at just the boxes.

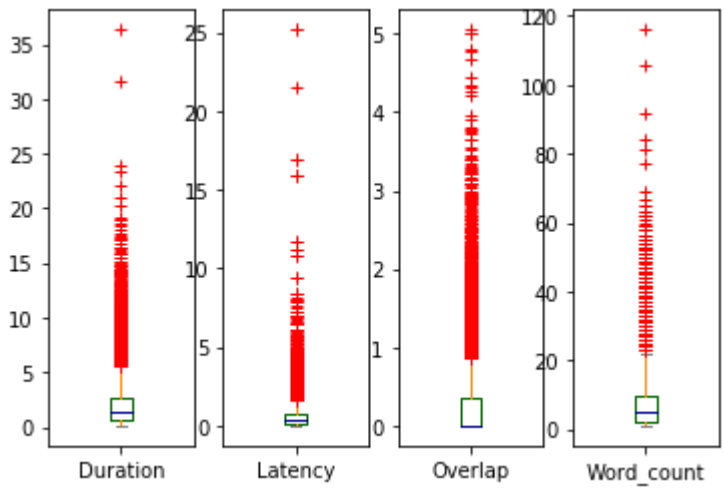
- For the Duration column, it's from 1 to 3.
- For the Latency column, it's around 0 to 1.
- For the Overlap column, it's from 0 to 0.5.
- For the word_count column, it's from 1 to 12.

I think the most significant difference lies in values outside the scope of the boxes; for the Latency and the Overlap column, the box value is similar, but for the Duration and the Word_count column, the values in the tagged data are in a more limited scope.

In [50]:

```
color = dict(boxes='DarkGreen', whiskers='DarkOrange',
             medians='DarkBlue', caps='Gray')

df_callhome_all[cols].plot(kind = 'box',subplots=True,color = color,sym='r+')
plt.show()
```



We also want to take a look at the comparison of NSU vs. SU（sentential utterances) vs. disfluency in the sample data

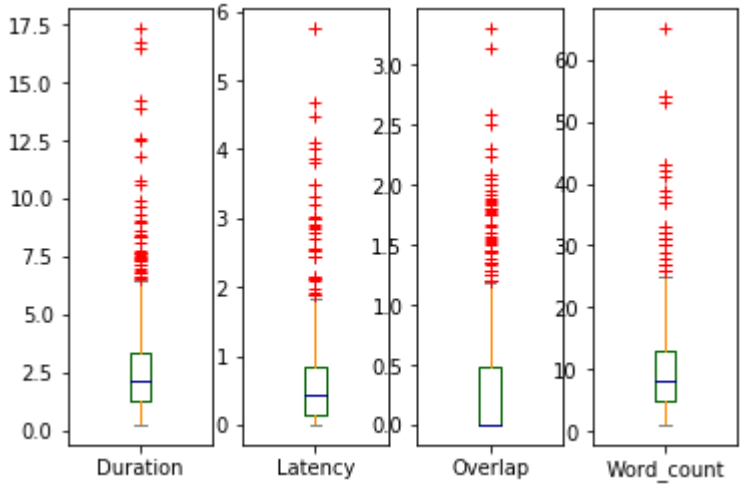
For the sample_su data, there are also many outliers. We look at the boxes. The values are similar to what we have seen in the aggregate data.

- For the Duration column, it's from 1 to 3.
- For the Latency column, it's around 0 to 1.
- For the Overlap column, it's from 0 to 0.5.
- For the word_count column, it's from 1 to 12.

In [51]:

```
color = dict(boxes='DarkGreen', whiskers='DarkOrange',
             medians='DarkBlue', caps='Gray')

call_su[cols].plot(kind = 'box',subplots=True,color = color,sym='r+')
plt.show()
```



For the sample_disfluency data, the boxes are rather clear and only a few outliers.

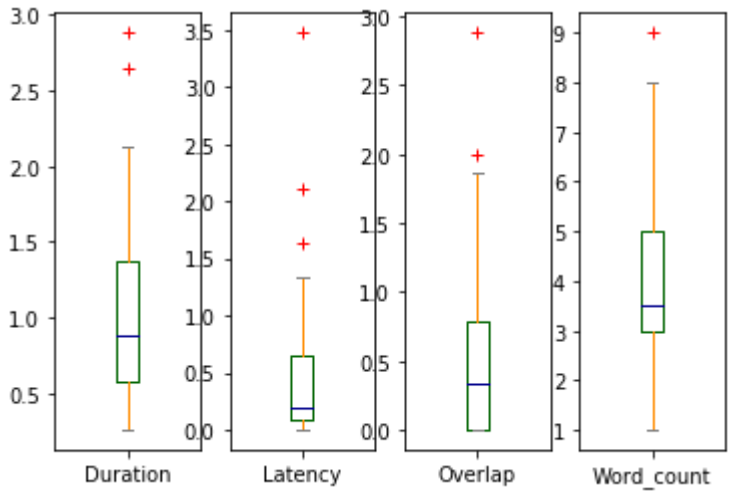
- For the Duration column, it's from 0.5 to 1.5.
- For the Latency column, it's around 0 to 0.7.
- For the Overlap column, it's from 0 to 0.8.
- For the word_count column, it's from 3 to 5.

As the disfluency can be confounded with NSU, we need to look at their critical differences, and according to this sample data, we think that the difference in the word_count is most evident.

In [52]:

```
color = dict(boxes='DarkGreen', whiskers='DarkOrange',
             medians='DarkBlue', caps='Gray')

call_disfluency[cols].plot(kind = 'box',subplots=True,color = color,sym='r+')
plt.show()
```



If we want to see the comparison of columns in the same figure and not by dataset.

4. Find the most frequent unigrams, bigrams and trigrams in the text and in the tags

Import the tagged data

In [56]:

```
TAGGED_FILE = 'Data/callhome_id_trans.csv.tag'

PATTERN_LINE_TAGGED = re.compile('(.+)\n')
```

Functions to read the tagged file

In [57]:

```
def readFile(filename,regexp,trans_group,silentregexp=None,verbose=False):
    '''
    IN :
    OUT:A corpus as a list of utterances non segmented yet = list of strings
    '''
    corpus = []
    if verbose:
        print(filename)
    i = 0
#    with codecs.open(filename,'r',encoding='utf8') as finput:
    finput = open(filename,'r',encoding='utf-8',errors='ignore')
    for line in finput:
        i += 1
        parsed = regexp.match(line)
        if parsed != None:
            transcription = parsed.group(trans_group)
            if silentregexp != None:
                transcription = re.sub(silentregexp,'',transcription)
            if transcription[0] !="#":
                corpus.append(transcription)
            if i == 1000000:
                break ;
        else:
            if line != "\n" and i<10 and verbose:
                print(line)
                i +=1
            if i == 100:
                break ;

    print('# Lines : {0:d}'.format(len(corpus)))

    return corpus
```

A function to look up the Correponding POS tags

In [58]:

```
def create_utt_lookup(tagged):
    lookup = {}
    for utt in tagged:
        key=''
        for n in re.findall(r'[\u4e00-\u9fff]+', utt):
            key = key+n
        if key not in lookup.keys():
            lookup[key] = utt
    return lookup
```

In [59]:

```
tagged = readFile(TAGGED_FILE,PATTERN_LINE_TAGGED,1)


lookup = create_utt_lookup(tagged)
```

```
# Lines : 32550
```

In [60]:

```
tagged
```

Out[60]:

```
['Text_NR',
 '你_PN 很_AD 爽_AD 哈_VV 你_PN ,_PU 哈_VV ?_PU',
 '要_VV 不_AD 要_VV 跟_P 妈_NN 讲话_VV ?_PU',
 '啊_VA ?_PU',
 '啊_VA ?_PU',
 '&乖_NR 乖&_VV ,_PU 哎_AD 小\_JJ &朱&_NR 实在_AD 是_VC 很_AD 有_VE 有_VE
福气_NN ._PU 他_PN 一直_AD 在_AD 盼_VV 着_AS 说_VV ,_PU 哎哟_NR 今天_NT
哥哥_NN 会_VV 不_AD 会_VV 打_VV 电话_NN 来_VV ,_PU 我_PN 有_VE 话_NN 跟_P
他_PN 讲_VV 喔_SP ,_PU 要_VV 不_AD 然_VV 明天_NT 到_VV 你_PN 办公室_NN 去
_MSP 花_VV 钱_NN 去_MSP 打_VV 好_VA 咯_VV ._PU',
 '哎_VV 妈妈_NN 啊_SP ._PU',
 '%呃_NN %_VV ._PU',
 '呃_JJ 呵_NN 呃_VV ,_PU 你_PN 看_VV 结果_NN 今天_NT 打_VV 这_DT 种_M 免
费_JJ 的_DEG ._PU',
 '结果_AD 你_PN 就_AD 打_VV 来_VV 了_AS ,_PU 他_PN 真_AD 有_VE ,_PU 这_D
T 小\_M &朱&_NR 就_AD 是_VC 没_VE 话_NN 说_VV ,_PU 就_AD 是_VC 有_VE 运气_
NN 的_DEC ,_PU 才_AD 讲_VV 的_DEC ,_PU 心_NN 里_LC 想_VV 的_DEC 就_AD ,_
PU 就_AD 打_VV 来_VV 了_AS ._PU',
```

In [61]:

```
type(tagged)
```

Out[61]:

```
list
```

Function to clean the original tagged data

In [62]:

```
def taggedrawtoclean(tagged_raw):
    res = []
    for line_tagged in tagged_raw:
        resline = []
        for t in line_tagged.split() :
            tlist = t.split('_')
            resline.append((tlist[0],tlist[1],tlist[0]))
        res.append(resline)
    return res
```

In [63]:

```
call_su_tagged = taggedrawtoclean(call_su['Tagged'])
```

In [64]:

```
call_disfluency_tagged = taggedrawtoclean(call_disfluency['Tagged'])
```

In [65]:

```
callhome_zh_tagged = taggedrawtoclean(tagged)
```

In [66]:

```
callhome_zh_tagged[0]
```

Out[66]:

```
[('Text', 'NR', 'Text')]
```

In [67]:

```
# Flatten list of list
call_tagged_tokens = [item for sublist in callhome_zh_tagged for item in sublist]
```

Look at the ngrams for words and tags in SU of the sample corpus

In [68]:

```
# Flatten list of list
call_su_tagged_tokens = [item for sublist in call_su_tagged for item in sublist]
```

In [69]:

```
len(call_su)
```

Out[69]:

```
951
```

In [70]:

```
call_su_tokens = [t[0] for t in call_su_tagged_tokens]
```

In [71]:

```
call_su_tokens_df = nltk.FreqDist(call_su_tokens)
for tok in [',','.','?','--','&']:
    call_su_tokens_df.pop(tok)
```

In [72]:

```
su_unigram_30 = call_su_tokens_df.most_common(30)
su_unigram_30
```

Out[72]:

```
[('我', 374),
 ('你', 357),
 ('个', 325),
 ('是', 285),
 ('的', 279),
 ('啊', 278),
 ('不', 266),
 ('那', 264),
 ('他', 229),
 ('就', 227),
 ('了', 211),
 ('这', 207),
 ('要', 189),
 ('一', 144),
 ('有', 138),
 ('好', 119),
 ('哎', 107),
 ('在', 103),
 ('说', 101),
 ('去', 94),
 ('还', 93),
 ('现在', 90),
 ('也', 86),
 ('呃', 80),
 ('呢', 78),
 ('什么', 72),
 ('都', 70),
 ('讲', 64),
 ('', 63),
 ('吧', 62)]
```

In [73]:

```
PUNCT = [',','.','?','--','(',')']

tokens = [x for x in call_su_tokens if x and x not in PUNCT]

bgs = nltk.bigrams(tokens)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(bgs)
fdist.most_common(20)
```

Out[73]:

```
[(('那', '个'), 129),
 (('这', '个'), 107),
 (('就', '是'), 83),
 (('不', '是'), 46),
 (('一', '个'), 45),
 (('不', '要'), 35),
 (('啊', '我'), 30),
 (('你', '你'), 28),
 (('还', '有'), 27),
 (('啊', '你'), 25),
 (('跟', '你'), 24),
 (('/', '/'), 24),
 (('你', '要'), 24),
 (('我', '觉得'), 24),
 (('你', '说'), 23),
 (('我', '就'), 23),
 (('家', '里'), 23),
 (('我', '跟'), 22),
 (('是', '说'), 22),
 (('不', '能'), 20)]
```

In [74]:

```
call_su_tags = [t[1] for t in call_su_tagged_tokens]
```

In [75]:

```
call_su_tags_df = nltk.FreqDist(call_su_tags)

call_su_tags_df = nltk.FreqDist(call_su_tags)
for tok in ['PU','']:
    call_su_tags_df.pop(tok)
```

In [76]:

```
call_su_tags_df.most_common(20)
```

Out[76]:

```
[('VV', 2153),
 ('AD', 1737),
 ('NN', 1412),
 ('PN', 1199),
 ('M', 599),
 ('DT', 467),
 ('VA', 422),
 ('NR', 398),
 ('CD', 322),
 ('P', 307),
 ('VC', 281),
 ('SP', 274),
 ('AS', 211),
 ('VE', 195),
 ('DEC', 184),
 ('NT', 183),
 ('JJ', 89),
 ('DEG', 87),
 ('LC', 81),
 ('CS', 33)]
```

Look at the ngrams for words and tags in disfluency of the sample corpus

In [77]:

```
# Flatten list of list
call_disfluency_tagged_tokens = [item for sublist in call_disfluency_tagged for item
```

In [78]:

```
call_disfluency_tokens = [t[0] for t in call_disfluency_tagged_tokens ]
```

In [79]:

```
call_disfluency_tokens_df = nltk.FreqDist(call_disfluency_tokens)
for tok in [',','.','?','--']:
    call_disfluency_tokens_df.pop(tok)
```

In [80]:

```
dis_unigram_10 = call_disfluency_tokens_df.most_common(10)
dis_unigram_10
```

Out[80]:

```
[('你', 15),
 ('这', 15),
 ('那', 12),
 ('就', 11),
 ('哎', 10),
 ('个', 10),
 ('我', 8),
 ('是', 7),
 ('啊', 5),
 ('他', 5)]
```

In [81]:

```
call_disfluency_tags = [t[1] for t in call_disfluency_tagged_tokens]
```

In [82]:

```
call_disfluency_tags_df = nltk.FreqDist(call_disfluency_tags)

call_disfluency_tags_df = nltk.FreqDist(call_disfluency_tags)
for tok in ['PU','']:
    call_disfluency_tags_df.pop(tok)
```

In [83]:

```
call_disfluency_tags_df.most_common(20)
```

Out[83]:

```
[('AD', 46),
 ('PN', 38),
 ('VV', 36),
 ('NN', 21),
 ('DT', 21),
 ('M', 17),
 ('P', 9),
 ('VA', 7),
 ('VC', 7),
 ('CD', 5),
 ('NR', 2),
 ('VE', 2),
 ('DEC', 1),
 ('>', 1),
 ('SP', 1),
 ('JJ', 1),
 ('NT', 1)]
```

Now we aim to find the most frequent n grams for words in the whold dataset

In [84]:

```
call_tokens = [t[0] for t in call_tagged_tokens]
```

In [85]:

```
call_tokens[0:10]
```

Out[85]:

```
['Text', '你', '很', '爽', '哈', '你', ',', '哈', '?', '要']
```

In [86]:

```
call_tokens_df = nltk.FreqDist(call_tokens)
for tok in [',','.','?','--','&']:
    call_tokens_df.pop(tok)
```

In [87]:

```
unigram_30 = call_tokens_df.most_common(30)
unigram_30
```

Out[87]:

```
[('我', 7970),
 ('是', 7931),
 ('那', 6761),
 ('的', 6759),
 ('你', 6708),
 ('个', 6535),
 ('就', 6315),
 ('啊', 5646),
 ('了', 5514),
 ('不', 5483),
 ('他', 4255),
 ('嗯', 3793),
 ('这', 3752),
 ('呃', 3542),
 ('对', 3370),
 ('说', 3089),
 ('哦', 3047),
 ('一', 3002),
 ('哎', 2756),
 ('好', 2609),
 ('要', 2433),
 ('有', 2249),
 ('也', 2167),
 ('还', 2151),
 ('在', 1935),
 ('什么', 1878),
 ('都', 1816),
 ('现在', 1792),
 ('吧', 1681),
 ('去', 1527)]
```

In [88]:

```
type(unigram_30)
```

Out[88]:

```
list
```

In [89]:

```
PUNCT = [',','.','?','--','(',')']

tokens = [x for x in call_tokens if x and x not in PUNCT]

bgs = nltk.bigrams(tokens)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(bgs)
fdist.most_common(20)
```

Out[89]:

```
[(('那', '个'), 3002),
 (('就', '是'), 2348),
 (('这', '个'), 1410),
 (('一', '个'), 1057),
 (('不', '是'), 914),
 (('对', '对'), 875),
 (('是', '说'), 795),
 (('嗯', '嗯'), 613),
 (('好', '的'), 537),
 (('我', '就'), 519),
 (('啊', '啊'), 483),
 (('哦', '哦'), 463),
 (('我', '我'), 459),
 (('你', '你'), 458),
 (('是', '吧'), 421),
 (('呃', '呃'), 415),
 (('不', '要'), 391),
 (('也', '不'), 380),
 (('的', '那'), 379),
 (('是', '不'), 373)]
```

In [90]:

```
PUNCT = [',','.','?','--','(',')','<English']

tokens = [x for x in call_tokens if x and x not in PUNCT]

tgs = nltk.trigrams(tokens)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(tgs)
fdist.most_common(20)
```

Out[90]:

```
[(('就', '是', '说'), 712),
 (('对', '对', '对'), 414),
 (('是', '不', '是'), 261),
 (('的', '那', '个'), 212),
 (('挺', '好', '的'), 210),
 (('个', '那', '个'), 203),
 (('那', '个', '那'), 201),
 (('&', '美国', '&'), 176),
 (('我', '跟', '你'), 173),
 (('有', '一', '个'), 165),
 (('是', '那', '个'), 162),
 (('嗯', '嗯', '嗯'), 152),
 (('哦', '哦', '哦'), 133),
 (('那', '个', '什么'), 126),
 (('&', '中国', '&'), 125),
 (('你', '那', '个'), 124),
 (('个', '就', '是'), 115),
 (('跟', '你', '说'), 113),
 (('啊', '啊', '啊'), 113),
 (('那', '个', '就'), 96)]
```

Now we turn to find the most frequent unigrams, bigrams and trigrams in the tags

In [62]:

```
call_tags = [t[1] for t in call_tagged_tokens]
```

In [63]:

```
call_tags_df = nltk.FreqDist(call_tags)

call_tags_df = nltk.FreqDist(call_tags)
for tok in ['PU','']:
    call_tags_df.pop(tok)
```

In [64]:

```
call_tags_df.most_common(20)
```

Out[64]:

```
[('VV', 47172),
 ('AD', 46021),
 ('NN', 31818),
 ('PN', 26361),
 ('VA', 13908),
 ('M', 12797),
 ('DT', 10420),
 ('P', 8495),
 ('NR', 8448),
 ('VC', 7828),
 ('CD', 7233),
 ('SP', 5347),
 ('AS', 5317),
 ('DEC', 4406),
 ('NT', 3965),
 ('VE', 3954),
 ('DEG', 2013),
 ('JJ', 1659),
 ('LC', 1510),
 ('CS', 482)]
```

In [65]:

```
stoptags = ['PU','']

tags = [x for x in call_tags if x and x not in stoptags]


bitags = nltk.bigrams(tags)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(bitags)
fdist.most_common(20)
```

Out[65]:

```
[(('AD', 'VV'), 15736),
 (('AD', 'AD'), 9104),
 (('PN', 'VV'), 8403),
 (('VV', 'VV'), 8216),
 (('NN', 'AD'), 7933),
 (('PN', 'AD'), 7471),
 (('VV', 'PN'), 7000),
 (('VV', 'AD'), 6743),
 (('DT', 'M'), 6569),
 (('VV', 'NN'), 6039),
 (('NN', 'VV'), 6014),
 (('CD', 'M'), 5481),
 (('M', 'NN'), 5201),
 (('AD', 'VA'), 5164),
 (('AD', 'VC'), 4586),
 (('VV', 'AS'), 4436),
 (('AD', 'PN'), 4202),
 (('VA', 'AD'), 3407),
 (('NR', 'NR'), 3052),
 (('NN', 'PN'), 2937)]
```

In [66]:

```
tags = [x for x in call_tags if x and x not in stoptags]

tritags = nltk.trigrams(tags)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(tritags)
fdist.most_common(20)
```

Out[66]:

```
[(('AD', 'AD', 'VV'), 3563),
 (('AD', 'VV', 'VV'), 3363),
 (('PN', 'AD', 'VV'), 3310),
 (('DT', 'M', 'NN'), 2748),
 (('VV', 'AD', 'VV'), 2547),
 (('NN', 'AD', 'VV'), 2431),
 (('AD', 'VV', 'AD'), 2346),
 (('AD', 'VV', 'PN'), 2179),
 (('VV', 'PN', 'VV'), 2058),
 (('CD', 'M', 'NN'), 2054),
 (('PN', 'AD', 'AD'), 1851),
 (('AD', 'VV', 'NN'), 1778),
 (('PN', 'VV', 'PN'), 1741),
 (('AD', 'VV', 'AS'), 1671),
 (('AD', 'AD', 'AD'), 1666),
 (('VV', 'PN', 'AD'), 1648),
 (('NN', 'AD', 'AD'), 1619),
 (('P', 'PN', 'VV'), 1582),
 (('AD', 'PN', 'AD'), 1467),
 (('VV', 'CD', 'M'), 1447)]
```

We will look at the POS-tags for the portion of data with NSU labels, so let's select this portion.

In [85]:

```
lookup = create_utt_lookup(tagged)

# Your data file (whatever it is)
corpus_dataframe = call_nsu
```

In [86]:

```
tagged_text = []
for utt in corpus_dataframe['Text']:
    key=''
    for n in re.findall(r'[\u4e00-\u9fff]+', utt):
        key = key+n
    if key in lookup.keys():
        tagged_text.append(lookup[key])
    else:
        tagged_text.append("%%%")

corpus_dataframe["Tagged"] = tagged_text
```

In [89]:

```
corpus_dataframe.head()
```

Out[89]:

	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	Start	Text	Word_
2	3	882	0.42	186.45	0.08	0.00	False	A	186.03	啊？	
3	4	882	0.28	187.17	0.44	0.00	False	B	186.89	啊？	
5	6	882	0.71	189.73	1.55	0.71	False	A	189.02	哎妈妈啊.	
6	7	882	0.73	197.09	NaN	NaN	True	A	196.36	%呃%.	
11	12	882	0.76	212.74	NaN	0.76	False	A	211.98	呃对，	

In [90]:

```
corpus_dataframe.to_csv('/Users/myday/Downloads/nb/nsutag_0518')
```

In [91]:

```
callhome_nsu_tagged = taggedrawtoclean(tagged_text)
```

In [92]:

```
callhome_nsu_tagged[0]
```

Out[92]:

```
[('啊', 'VA', '啊'), ('?', 'PU', '?')]
```

In [93]:

```
# Flatten list of list
nsu_tagged_tokens = [item for sublist in callhome_nsu_tagged for item in sublist]
```

In [94]:

```
nsu_tagged_tokens
```

Out[94]:

```
[('啊', 'VA', '啊'),
 ('?', 'PU', '?'),
 ('啊', 'VA', '啊'),
 ('?', 'PU', '?'),
 ('哎', 'VV', '哎'),
 ('妈妈', 'NN', '妈妈'),
 ('啊', 'SP', '啊'),
 ('.', 'PU', '.'),
 ('%呃', 'NN', '%呃'),
 ('%', 'VV', '%'),
 ('.', 'PU', '.'),
 ('呃', 'AD', '呃'),
 ('对', 'VA', '对'),
 (',', 'PU', ','),
 ('呃', 'NN', '呃'),
 ('没', 'VE', '没'),
 ('问题', 'NN', '问题'),
 ('.', 'PU', '.'),
```

Now for the NSU labeled corpus portion, we find the most frequent unigrams, bigrams and trigrams in the words

In [74]:

```
nsu_tokens = [t[0] for t in nsu_tagged_tokens]
```

In [75]:

```
nsu_tokens[0:10]
```

Out[75]:

```
['啊', '?', '啊', '?', '哎', '妈妈', '啊', '.', '%呃', '%']
```

In [76]:

```
nsu_tokens_df = nltk.FreqDist(nsu_tokens)
for tok in [',','.','?','&']:
    nsu_tokens_df.pop(tok)
```

In [77]:

```
nsu_unigram_30 = nsu_tokens_df.most_common(30)
nsu_unigram_30
```

Out[77]:

```
[('嗯', 144),
 ('啊', 92),
 ('哎', 89),
 ('对', 55),
 ('哦', 55),
 ('%呃', 32),
 ('%', 32),
 ('是', 32),
 ('的', 28),
 ('呃', 27),
 ('好', 25),
 ('了', 24),
 ('不', 22),
 ('呀', 22),
 ('吧', 14),
 ('啦', 13),
 ('哼', 13),
 ('那', 11),
 ('还', 9),
 ('钱', 8),
 ('就', 8),
 ('没', 7),
 ('这样', 7),
 ('知道', 7),
 ('哈', 7),
 ('你', 7),
 ('会', 7),
 ('一', 7),
 ('个', 7),
 ('得', 7)]
```

In [78]:

```
PUNCT = [',','.','?','--','(',')']

tokens = [x for x in nsu_tokens if x and x not in PUNCT]

nsu_bgs = nltk.bigrams(tokens)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(nsu_bgs)
fdist.most_common(20)
```

Out[78]:

```
[(('嗯', '嗯'), 55),
 (('%呃', '%'), 32),
 (('哎', '哎'), 26),
 (('哦', '哦'), 21),
 (('对', '对'), 17),
 (('啊', '啊'), 14),
 (('嗯', '哎'), 13),
 (('嗯', '哦'), 11),
 (('哎', '嗯'), 11),
 (('%', '%呃'), 10),
 (('啊', '嗯'), 10),
 (('哎', '对'), 9),
 (('啊', '哎'), 8),
 (('嗯', '哼'), 8),
 (('的', '啊'), 8),
 (('哎', '是'), 8),
 (('啊', '%呃'), 7),
 (('呃', '呃'), 7),
 (('对', '啊'), 7),
 (('对', '了'), 7)]
```

In [79]:

```
PUNCT = [',','.','?','--','(',')','<English']

tokens = [x for x in nsu_tokens if x and x not in PUNCT]

nsu_tgs = nltk.trigrams(tokens)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(nsu_tgs)
fdist.most_common(20)
```

Out[79]:

```
[(('嗯', '嗯', '嗯'), 23),
 (('哦', '哦', '哦'), 13),
 (('哎', '哎', '哎'), 11),
 (('%呃', '%', '%呃'), 10),
 (('%', '%呃', '%'), 10),
 (('啊', '%呃', '%'), 7),
 (('对', '对', '对'), 6),
 (('%呃', '%', '对'), 5),
 (('真', '的', '啊'), 5),
 (('&绿', '酶素', '&'), 5),
 (('呃', '呃', '呃'), 4),
 (('不', '会', '啦'), 4),
 (('哎', '哎', '嗯'), 4),
 (('嗯', '哎', '哎'), 4),
 (('哎', '哎', '对'), 4),
 (('哎', '对', '了'), 4),
 (('嗯', '嗯', '哎'), 4),
 (('哎', '%呃', '%'), 4),
 (('%呃', '%', '哎'), 4),
 (('哎', '嗯', '嗯'), 4)]
```

Now we turn to find the most frequent unigrams, bigrams and trigrams in the tags

In [80]:

```
nsu_tags = [t[1] for t in nsu_tagged_tokens]
```

In [81]:

```
nsu_tags_df = nltk.FreqDist(nsu_tags)

nsu_tags_df = nltk.FreqDist(nsu_tags)
for tok in ['PU','']:
    nsu_tags_df.pop(tok)
```

In [82]:

```
nsu_tags_df.most_common(20)
```

Out[82]:

```
[('AD', 234),
 ('VA', 221),
 ('VV', 191),
 ('NN', 183),
 ('NR', 51),
 ('P', 47),
 ('CD', 37),
 ('SP', 35),
 ('M', 33),
 ('VC', 31),
 ('PN', 26),
 ('NT', 26),
 ('AS', 22),
 ('DEC', 17),
 ('FW', 14),
 ('DT', 13),
 ('VE', 11),
 ('IJ', 10),
 ('OD', 6),
 ('DEG', 6)]
```

In [83]:

```
stoptags = ['PU','']

tags = [x for x in nsu_tags if x and x not in stoptags]


nsu_bitags = nltk.bigrams(tags)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(nsu_bitags)
fdist.most_common(20)
```

Out[83]:

```
[(('VA', 'AD'), 61),
 (('AD', 'AD'), 59),
 (('NN', 'VV'), 58),
 (('AD', 'VA'), 57),
 (('AD', 'VV'), 55),
 (('VA', 'VA'), 49),
 (('NN', 'VA'), 42),
 (('VV', 'NN'), 37),
 (('VV', 'AD'), 34),
 (('VA', 'NN'), 29),
 (('NN', 'AD'), 27),
 (('CD', 'M'), 26),
 (('VV', 'VV'), 26),
 (('VV', 'VA'), 24),
 (('VV', 'AS'), 15),
 (('VA', 'VV'), 14),
 (('NN', 'NN'), 14),
 (('P', 'NN'), 14),
 (('P', 'P'), 14),
 (('VA', 'DEC'), 13)]
```

In [84]:

```
tags = [x for x in nsu_tags if x and x not in stoptags]

nsu_tritags = nltk.trigrams(tags)

#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(nsu_tritags)
fdist.most_common(20)
```

Out[84]:

```
[(('AD', 'VA', 'AD'), 19),
 (('NN', 'VV', 'NN'), 17),
 (('VA', 'AD', 'VA'), 17),
 (('AD', 'AD', 'VA'), 16),
 (('VV', 'NN', 'VV'), 16),
 (('AD', 'AD', 'VV'), 16),
 (('AD', 'AD', 'AD'), 16),
 (('VA', 'VA', 'VA'), 16),
 (('NN', 'VV', 'AD'), 15),
 (('AD', 'VV', 'VV'), 14),
 (('VA', 'NN', 'VV'), 12),
 (('VA', 'NN', 'VA'), 12),
 (('VA', 'VA', 'AD'), 12),
 (('VV', 'AD', 'VV'), 11),
 (('NN', 'VA', 'NN'), 10),
 (('VA', 'AD', 'VV'), 10),
 (('NN', 'VA', 'AD'), 10),
 (('VA', 'AD', 'AD'), 10),
 (('VV', 'AD', 'AD'), 9),
 (('VA', 'CD', 'M'), 9)]
```

In [55]:

```
lookup = create_utt_lookup(tagged)

# Your data file (whatever it is)
corpus_dataframe = call_su
```

In [56]:

```
tagged_text = []
for utt in corpus_dataframe['Text']:
    key=''
    for n in re.findall(r'[\u4e00-\u9fff]+', utt):
        key = key+n
    if key in lookup.keys():
        tagged_text.append(lookup[key])
    else:
        tagged_text.append("%%%")

corpus_dataframe["Tagged"] = tagged_text
```

```
/Users/myday/snorkel-tutorials/.envspam/lib/python3.7/site-packages/ip
ykernel_launcher.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  # This is added back by InteractiveShellApp.init_path()
```

In [57]:

```
corpus_dataframe.head()
```

Out[57]:

	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	Start	Text	Word
0	1	882	1.73	185.20	NaN	NaN	False	A	183.47	你很爽哈你，哈？	
1	2	882	1.02	185.95	NaN	0.27	False	B	184.93	要不要跟妈讲话？	
4	5	882	10.78	199.50	1.55	0.00	False	B1	188.72	&乖乖&，哎小&朱&实在是很有有福气.他一直在盼着说，...	
7	8	882	2.88	201.64	NaN	NaN	True	A	198.76	{laugh}呃呵呃，你看结果今天打这种免费的.	
8	9	882	6.52	206.61	NaN	1.55	False	B1	200.09	结果你就打来了，他真有，这小&朱&就是没话说，就是有...	

In [58]:

```
corpus_dataframe.to_csv('/Users/myday/Downloads/nb/sutag_0518')
```

In [59]:

```
lookup = create_utt_lookup(tagged)

# Your data file (whatever it is)
corpus_dataframe = call_disfluency
```

In [60]:

```
tagged_text = []
for utt in corpus_dataframe['Text']:
    key=''
    for n in re.findall(r'[\u4e00-\u9fff]+', utt):
        key = key+n
    if key in lookup.keys():
        tagged_text.append(lookup[key])
    else:
        tagged_text.append("%%%")

corpus_dataframe["Tagged"] = tagged_text
```

```
/Users/myday/snorkel-tutorials/.envspam/lib/python3.7/site-packages/ip
ykernel_launcher.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  # This is added back by InteractiveShellApp.init_path()
```

In [61]:

```
corpus_dataframe.head()
```

Out[61]:

	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	Start	Text	Wo
26	27	882	2.00	254.39	0.23	2.00	False	B1	252.39	通知你的啊？他知道你，	
50	51	882	0.73	307.61	0.49	0.73	False	B	306.88	可是去你，	
65	66	882	0.96	333.61	0.07	0.96	False	A	332.65	那那你怎么(())，	
112	113	882	0.42	436.09	NaN	0.42	False	A	435.67	礼拜-	
171	172	1711	0.44	248.88	0.42	0.00	False	A	248.44	那特简-	

Summary

We have seen the descriptive statistics of our corpus (including the sample, which constitutes 5% of the whole data. The distribution plots help us to see the distribution of each column. The pairplots indicate some correlation between the columns. The comparison of NSU labeled data, and other portions of data can serve as distinguishing cues, especially for the numerical columns.

In this notebook, the aim is to build a model of detecting NSUs in our corpus CallHome using Snorkel.

Brief introduction about the Snorkel

Using Snorkel, it will train a model that combines all the rules defined written to estimate their accuracy, along with the overlaps and conflicts among different labeling functions.

The use of Snorkel includes writing diverse labeling functions(LF) to produce some useful training data with labels. A labeling function is a rule that attributes a label for some subset of the training dataset.

Labeling functions do not need to be entirely accurate or exhaustive and can even be correlated. Snorkel will automatically estimate their accuracies and correlations [in a provably consistent way (https://papers.nips.cc/paper/6523-data-programming-creating-large-training-sets-quickly)](https://papers.nips.cc/paper/6523-data-programming-creating-large-training-sets-quickly). Several strategies to write labeling functions: keyword matches, regular expressions, arbitrary heuristics, third-party models.

Brief introduction about the features that we consider useful in writing the labeling functions.

We selected the first 1500 lines of the whole corpus (5% of the corpus) and labeled the data manually. We could use the differences between the data with the NSU tag and the complete data to write some Labelling functions to build a model.

The numerical columns are useful, especially the "Duration" and "Word_count" column, where the difference is the most evident.

Also, we will use other features like some keywords in the "Text" column, as we have done in the **Descriptive Statistics Notebook**, we may attend to the frequent words(patterns) with NSU tags; also we may use the most common unigrams, bigrams and trigrams in words and POS tags.

0. Set the environment and import the needed modules

In [1]:

```
#%matplotlib inline

import os

# Make sure we're running from the spam/ directory
if os.path.basename(os.getcwd()) == "snorkel-tutorials":
    os.chdir("callhome")

# Turn off TensorFlow logging messages
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# For reproducibility
os.environ["PYTHONHASHSEED"] = "0"
```

In [2]:

```
import os
import re
import string
import glob
import nltk
from collections import defaultdict
from collections import Counter


# Computing / Stats packages
import pandas as pd
import numpy as np

# Classification
from sklearn.feature_extraction.text import CountVectorizer


# Visualisation
import seaborn as sns
import matplotlib.pyplot as plt

# Snorkel
import snorkel
from snorkel.labeling import LabelingFunction
from snorkel.labeling import PandasLFApplier
from snorkel.labeling import LFAnalysis
from snorkel.labeling import MajorityLabelVoter
from snorkel.labeling import LabelModel
from snorkel.labeling import filter_unlabeled_dataframe

from snorkel.analysis import get_label_buckets


DISPLAY_ALL_TEXT = False

pd.set_option("display.max_colwidth", 0 if DISPLAY_ALL_TEXT else 50)
```

1.Read the CallHome corpus

In [3]:

```
df_callhome_all = pd.read_csv('Data/calltags_0519.csv')
```

In [4]:

```
df_callhome_all.head()
```

Out[4]:

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	Start	T
0	0	1	882	1.73	185.20	NaN	NaN	False	A	183.47	
1	1	2	882	1.02	185.95	NaN	0.27	False	B	184.93	
2	2	3	882	0.42	186.45	0.08	0.00	False	A	186.03	
3	3	4	882	0.28	187.17	0.44	0.00	False	B	186.89	
4	4	5	882	10.78	199.50	1.55	0.00	False	B1	188.72	

Restatement(in the *descriptive statistics notebook*) of the pilot study with 5% of the corpus)

We also select around 5% of the whole data(the first 1500 lines of the entire information) to tag the data manually to have an idea about the difference between data with NSU tags and the whole data. We expect that the numerical columns may behave differently. Because the NSUs are generally short utterances so the duration might be brief, and the counted words may be less than the mean of all the data. Same for the Latency and overlap, so we want to know the exact difference. Below is the results.

For the use of this detection task, all the NSUs are labeled as 1, all the Non-NSUs are labeled as 0.

In [5]:

```
callhome_labelled = pd.read_csv('Data/label_0523.csv')
```

In [6]:

```
callhome_labelled.head()
```

Out[6]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Latency	Ov
0	1	882	183.47	185.20	A	你很爽哈你，哈？	6	1.73	False	NaN	
1	2	882	184.93	185.95	B	要不要跟妈讲话？	5	1.02	False	NaN	
2	3	882	186.03	186.45	A	啊？	1	0.42	False	0.08	
3	4	882	186.89	187.17	B	啊？	1	0.28	False	0.44	
4	5	882	188.72	199.50	B1	&乖乖&，哎小&朱&实在是很有有福气.他一直在盼着说，...	42	10.78	False	1.55	

In [7]:

```
call1500=callhome_labelled
len(call1500)
```

Out[7]:

```
1501
```

Extract the data with NSU labels; this time, we assign the non-NSU to 0, so it's all the non zero labels. It should be noted that a non-NSU type"disfluency" are labeled 30, we give it a label because they are generally short utterances, if we only look at the duration and Word_count, they are easily confounded with the NSU. Having this class is useful for our task of detection of NSU, but now we just look at the NSU data.

In [8]:

```
call_nsu = call1500[call1500["Label"]!= 0]
```

In [9]:

```
len(call_nsu)
```

Out[9]:

```
486
```

Example of Comparaison of the distributions of the numerical columns of the sample nsu data and the original data

As we have presented in the **Descriptive statistics notebook**, we just look at the two columns "Duration" and "Word_count" where the difference is the most evident.

In [10]:

```
cols = ['Duration','Latency','Overlap','Word_count']

df_callhome_all[cols]['Duration'].mean()
```

Out[10]:

```
1.9155431921512325
```

In [11]:

```
call_nsu[cols]['Duration'].mean()
```

Out[11]:

```
0.7226748971193416
```

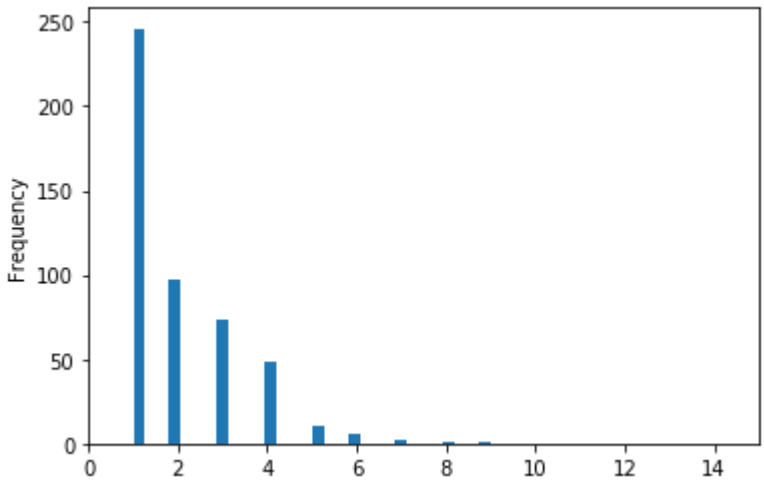
A closer look at the Word_count column, the most prominent value is one and the value from 2 to 6 is declining gradually, a few vlaue from 7 to 9.

In [12]:

```
call_nsu["Word_count"].plot(kind = 'hist',bins  =30,xlim = (0,15))
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x135ee4fd0>
```



We load the CallHome dataset and create Pandas DataFrame objects for the development, validation, and test sets.

Each DataFrame consists of the following fields:

- **`Start`** : Start time of the utterance sequence
- **`Text`** : Conversation transcription content
- **`Word_count`** : How many words(punctuations counted separately) in this segment
- **`Duration`** : How many seconds this segment lasts
- **`Overlap`** : The section when more than one speaker speaking at the same time
- **`Latency`** : The pause when the previous speaker has finished but the other speaker hasn't begin
- **`Same_speaker`** : If the two continous sequence is produced by the same speaker
- **`Label`** : Whether the comment is `NSU` (1), `SU(Sentential Utterance)` (0), or `UNKNOWN/ABSTAIN` (-1)

We start by loading our data. We have a manually labeled dataset(call1500) containing the first 1500 lines from the whole CallHome dataframe.

We split the call1500 data into dev, test, valid set by 6:2:2.

In [13]:

```
df_train = df_callhome_all[1500:]

#df_dev = call1500[0:int(len(call1500)*0.8)]
#df_test = call1500[int(len(call1500)*0.8):len(call1500)]



df_dev, df_test, df_valid = np.split(call1500.sample(frac=1), [int(.6*len(call1500))

print("Train Relationships: ", len(df_train))
print("Dev Relationships: ", len(df_dev))
print("Test Relationships: ", len(df_test))
print("Valid Relationships: ", len(df_valid))

Y_dev = df_dev.Label.values
Y_test = df_test.Label.values
Y_valid = df_valid.Label.values
```

```
Train Relationships:  31932
Dev Relationships:  900
Test Relationships:  300
Valid Relationships:  301
```

Let's view 5 data points from the dev set.

In [14]:

```
df_dev.sample(5, random_state=3)
```

Out[14]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	I
1340	1341	963	215.00	215.26	A1	{laugh}	1	0.26	False	
584	585	860	552.80	556.26	A	我 不 知道 他 有 没 这 个 , 我 , 可 , 可能 我 在 这个 , 胡扯 , 在 猜测 ,	16	3.46	False	
790	791	799	287.90	291.11	B	((什么)) [[distortion]] 颜色 的 ? 什么 颜色 ? 哎 .	7	3.21	False	
547	548	860	491.81	492.06	B	哎 .	1	0.25	True	
708	709	799	171.55	172.90	A	你 到 他 家 里 去 一次 啊 .	7	1.35	False	

In [15]:

```
# For clarity, we define constants to represent the class labels for NSU, Non-NSU, a
ABSTAIN = -1
SU = 0
NSU = 1

print(f"Dev NSU frequency: {100 * (df_dev.Label.values == NSU).mean():.1f}%")
```

```
Dev NSU frequency: 32.4%
```

2. Writing labeling functions

Based on the four columns of numerical indices, we define the Non-NSUs functions

a) Exploring the development set for initial ideas

We can start by looking at 20 random data points from the `train` set to generate some ideas for LFs.

In [16]:

```
df_train[["Duration", "Text", "Word_count"]].sample(20, random_state=2)
```

Out[16]:

	Duration	Text	Word_count
28987	0.61	就 在 吃 .	3
16359	0.56	是 哇 , 是 哇 .	4
10894	0.29	呃 ,	1
28564	0.25	[noise]	1
12236	0.41	呃 哼 .	2
1694	0.67	你 爸 在	3
33224	0.82	特别 恐怖 .	2
23802	1.46	对 , 那天 我 ((又)) 听 ((收音机)) 说 ,	7
29365	0.33	哦 ,	1
1893	3.33	哎 , 一定 要 注意 , 这 盐 稍微 要 控制 一点 , {laugh}	11
28329	4.71	我 找 工作 上面 的 也是 也是 说 , 就说 你 到 最后 叫 背水一战 的 时候 , ...	19
19829	2.50	啊 , 那 行 , 嗯 , 好 .	5
7762	1.34	((呃 对 , 在 &桂林& ,))	5
8773	3.47	那个 厂 卖给 这个 这 这个 药酒 制药厂 了 .	9
3026	0.50	嗯 哼 ,	2
1758	1.55	逮到 你 有空 的 时候 啊 , 呃 ?	7
26157	4.70	哎 ((详细 详细)) , 哎 有人 感兴趣 , 对 对 . 对 对 , 嗯 好 嘛 .	13
1500	1.10	我们 都 着急 # 死 了 #	7
33272	0.94	你 去 &泰国& 了 么 ?	5
23675	3.17	-- 还 没 洗 呢 , 因为 好 , 还有 好 几 张 照片 没 照 完 呢 .	16

As we have done some data explorations about our corpus in another notebook, we will go directly into using some indices we are rather sure to write the LFs.

LF about Cases when it's not an NSU. The numbers are decided based on the results in the *Descriptive Statistics Notebook*.

In [17]:

```
from snorkel.labeling import labeling_function


@labeling_function()
def duration(x):
    return SU if x['Duration']  > 4  else ABSTAIN

@labeling_function()
def latency(x):
    return SU if x['Latency']  > 4 else ABSTAIN


@labeling_function()
def overlap(x):
    return SU if x['Overlap']  > 2 else ABSTAIN

@labeling_function()
def wordcount(x):
    return SU if x['Word_count']  > 7 else ABSTAIN
```

Cases when it's probably an NSU.

In [18]:

```
@labeling_function()
def yduration(x):
    return NSU if x['Duration']  < 0.8  else ABSTAIN

@labeling_function()
def ywordcount(x):
    return NSU if x['Word_count']  in [1,2,3] else ABSTAIN
```

To apply one or more LFs that we've written to a collection of data points, we use an `LFApplier` (https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.LFApplier.html). Because our data points are represented with a Pandas DataFrame in this tutorial, we use the `PandasLFApplier` (https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.PandasLFApplier.ht Correspondingly, a single data point `x` that's passed into our LFs will be a Pandas `Series` object (https://pandas.pydata.org/pandas-docs/stable/reference/series.html).

It's important to note that these LFs will work for any object with an attribute named `text`, not just Pandas objects. Snorkel has several other appliers for different data point collection types which you can browse in the API documentation (https://snorkel.readthedocs.io/en/master/packages/labeling.html).

The output of the `apply(...)` method is a ***label matrix***, a fundamental concept in Snorkel. It's a NumPy array `L` with one column for each LF and one row for each data point, where `L[i, j]` is the label that the `j` th labeling function output for the `i` th data point. We'll create one label matrix for the `train` set and one for the `dev` set.

In [19]:

```
lfs = [duration, wordcount, yduration, ywordcount]

applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df=df_train)
L_dev = applier.apply(df=df_dev)
```

```
100%|██████████| 31932/31932 [00:01<00:00, 21943.16it/s]
100%|██████████| 900/900 [00:00<00:00, 21543.99it/s]
```

In [20]:

```
L_train
```

Out[20]:

```
array([[-1, -1, -1, -1],
       [-1,  0, -1, -1],
       [-1, -1,  1,  1],
       ...,
       [-1, -1, -1, -1],
       [-1, -1, -1,  1],
       [-1, -1, -1, -1]])
```

b) Evaluate performance on training and development sets

In [21]:

```
coverage_duration, coverage_wordcount,coverage_yduration, coverage_ywordcount = (L_t
print(f"duration coverage: {coverage_duration * 100:.1f}%")
print(f"wordcount coverage: {coverage_wordcount * 100:.1f}%")
print(f"yduration coverage: {coverage_yduration * 100:.1f}%")
print(f"ywordcount coverage: {coverage_ywordcount * 100:.1f}%")
```

```
duration coverage: 11.1%
wordcount coverage: 33.4%
yduration coverage: 33.6%
ywordcount coverage: 41.6%
```

Lots of statistics about labeling functions — like coverage — are useful when building any Snorkel application. So Snorkel provides tooling for common LF analyses using the `LFAnalysis` utility (https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.LFAnalysis.html). We report the following summary statistics for multiple LFs at once:

- **Polarity**: The set of unique labels this LF outputs (excluding abstains)
- **Coverage**: The fraction of the dataset the LF labels
- **Overlaps**: The fraction of the dataset where this LF and at least one other LF label
- **Conflicts**: The fraction of the dataset where this LF and at least one other LF label and disagree
- **Correct**: The number of data points this LF labels correctly (if gold labels are provided)
- **Incorrect**: The number of data points this LF labels incorrectly (if gold labels are provided)
- **Empirical Accuracy**: The empirical accuracy of this LF (if gold labels are provided)

For *Correct*, *Incorrect*, and *Empirical Accuracy*, we don't want to penalize the LF for data points where it abstained. We calculate these statistics only over those data points where the LF output a label. Since we have labels for the `dev` set but not the `train` set, we'll compute these statistics for the `dev` set only by supplying `Y_dev`.

In [22]:

```
LFAnalysis(L=L_train, lfs=lfs).lf_summary()
```

Out[22]:

	j	Polarity	Coverage	Overlaps	Conflicts
duration	0	[0]	0.110610	0.109514	0.000752
wordcount	1	[0]	0.333646	0.108856	0.000094
yduration	2	[1]	0.335588	0.318583	0.000094
ywordcount	3	[1]	0.416447	0.319241	0.000752

In [23]:

```
LFAnalysis(L=L_dev, lfs=lfs).lf_summary(Y=Y_dev)
```

Out[23]:

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
duration	0	[0]	0.095556	0.095556	0.001111	86	0	1.000000
wordcount	1	[0]	0.335556	0.094444	0.000000	301	1	0.996689
yduration	2	[1]	0.296667	0.273333	0.000000	208	59	0.779026
ywordcount	3	[1]	0.366667	0.274444	0.001111	251	79	0.760606

The helper method `get_label_buckets(...)` groups data points by their predicted label and true label. For example, we may find the indices of data points that the LF labeled `NSU` that actually belong to class `SU`. This may give ideas for where the LF could be made more specific.

For each LF, if we want to check the incorrect instances, we can use its j value to locate the ndarray of this exact functions. Like below we print the incorrect instances of the (j = 2) of the LF y_duration. We can see that the cases with limited short duration also encompass a lot of other cases than NSU, we will make some other LFs based on these points to improve the accuracy of our final model.

In [24]:

```
buckets = get_label_buckets(Y_dev, L_dev[:, 2])
df_dev.iloc[buckets[(SU,NSU)]]
```

Out[24]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Latency
668	669	860	703.48	703.98	A	就是 给 --	3	0.50	False	1.63
544	545	860	486.35	487.02	A	哦 , 我 抓 哎 .	4	0.67	False	0.19
622	623	860	618.64	619.33	A	我 跟 你 讲 , 他 跟 --	7	0.69	False	NaN
112	113	882	435.67	436.09	A	礼拜-	1	0.42	False	NaN

From the results above, we make these observations:

The utterances in English are always tagged as "Text_NR"; we can label all these cases as SU. It should be noted that the utterances marked as SU (0) do not mean they are all sentences in a linguistic sense; first, they may include disfluency cases. Second, the label is decided to conform to the purpose of the task, to select the NSU among others in a corpus.

Apart from "VV" who are not supposed to be in NSU, there are other words with their POS tags, which occur more often, such as "了_AS","要_VV", "是_VC".

The same analysis could be done for other LFs such as "ywordcount".

In [25]:

```
buckets = get_label_buckets(Y_dev, L_dev[:, 3])
df_dev.iloc[buckets[(SU,NSU)]]
```

Out[25]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Latency
668	669	860	703.48	703.98	A	就是 给 --	3	0.50	False	1.63
112	113	882	435.67	436.09	A	礼拜-	1	0.42	False	NaN
615	616	860	609.71	610.03	A	我 就 --	3	0.32	False	0.10
886	887	799	445.88	447.07	A	哎 , 叫 +富得宁+ .	3	1.19	False	0.11

We didn't compare one by one, but there are a lot of identical cases as in LF "yduration", which is easy to understand considering the correlation of "Duration" and "Word_count". From the numbers, we can see the mislabeled cases in this LFs include the last one, so we need to tell the system that some cases are not NSU by adding some relative LF.

In [26]:

```
@labeling_function()
def english(x):
    return SU if x['Tagged']  == "Text_NR"  else ABSTAIN
```

These two cases are actually NSU but with extra word_count.

Now let's take a look at 10 random `train` set data points where `duration` labeled `SU` to see if it matches our intuition or if we can identify some false positives.

In [27]:

```
df_train.iloc[L_train[:, 1] == SU].sample(10, random_state=1)
```

Out[27]:

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker
25117	25133	25134	952	4.44	483.96	NaN	NaN	True	A
22961	22976	22977	1582	2.34	230.90	NaN	NaN	False	A
23166	23181	23182	1582	2.13	548.04	NaN	NaN	True	A
12835	12846	12847	975	2.18	135.78	NaN	NaN	True	A1

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker
15089	15100	15101	669	7.77	629.99	0.64	0.00	False	A
2736	2736	2737	766	1.73	517.03	0.44	0.00	False	A1
32186	32236	32237	846	2.97	168.11	0.13	0.57	False	B
2118	2118	2119	1280	3.01	686.45	NaN	NaN	True	A1
20129	20142	20143	698	1.23	276.20	NaN	NaN	False	B

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker
30612	30661	30662	738	2.68	444.18	NaN	NaN	True	A

The results above shows no mislabeled case, the simple rules are performing good enough. Let's see 10 data points where `word_count` abstained, but `duration` labeled.

In [28]:

```
buckets = get_label_buckets(L_train[:, 0], L_train[:, 1])
df_train.iloc[buckets[(ABSTAIN, SU)]].sample(10, random_state=1)
```

Out[28]:

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker
23054	23069	23070	1582	2.35	374.25	NaN	NaN	True	A
28619	28665	28666	848	2.82	499.83	NaN	NaN	True	A
28280	28326	28327	840	3.76	460.26	0.58	0.00	False	B
19012	19025	19026	764	1.43	276.01	NaN	0.04	False	A

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker
4508	4509	4510	695	1.40	474.50	NaN	0.29	False	A
7598	7607	7608	760	3.17	322.91	0.65	0.00	False	B1
5988	5994	5995	704	1.57	161.34	0.04	0.84	False	A
2708	2708	2709	766	3.73	477.54	0.07	0.35	False	B
10492	10502	10503	861	2.35	604.10	0.69	0.33	False	B

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker
2757	2757	2758	766	3.09	549.32	NaN	NaN	True	A1

It shows one case when it's mislabeld by the LF duration. 5988 5994 5995 704 1.57 161.34 0.04 0.84 False A 159.77 嗯 , 嗯 , 嗯 , 嗯 , 嗯 , 右边 儿 的 .

c) Balance accuracy and coverage

We see from the bucket results there are some false positives and most of them have 1 word count, so we set another LF.

We also use some of the disfluency index to rule out some counfounds.

In [29]:

```
@labeling_function()
def wordcount_disfluency(x):
    return SU if x['Word_count'] in [3,4,5] else ABSTAIN
```

In [30]:

```
@labeling_function()
def duration_disfluency(x):
    return SU if (x['Duration'] > 0.5) and (x['Duration'] < 1.5) else ABSTAIN
```

In [31]:

```
@labeling_function()


def pos_dis(x):
    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    count = 0
    for tag in tags:
        count +=1
    if count < 4:
        if tag in ['VV','VE','AD','NR']:
            return SU
    return ABSTAIN
```

Again, let's generate our label matrices and see how we do.

In [32]:

```
lfs = [wordcount_disfluency,duration_disfluency, pos_dis]

applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df=df_train)
L_dev = applier.apply(df=df_dev)
```

```
100%|██████████| 31932/31932 [00:01<00:00, 21477.43it/s]
100%|██████████| 900/900 [00:00<00:00, 19867.55it/s]
```

In [33]:

```
LFAnalysis(L=L_train, lfs=lfs).lf_summary()
```

Out[33]:

	j	Polarity	Coverage	Overlaps	Conflicts
wordcount_disfluency	0	[0]	0.223036	0.169047	0.0
duration_disfluency	1	[0]	0.325160	0.188964	0.0
pos_dis	2	[0]	0.048697	0.027559	0.0

In [34]:

```
LFAnalysis(L_dev, lfs).lf_summary(Y=Y_dev)
```

Out[34]:

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
wordcount_disfluency	0	[0]	0.271111	0.214444	0.0	160	84	0.655738
duration_disfluency	1	[0]	0.376667	0.215556	0.0	206	133	0.607670
pos_dis	2	[0]	0.022222	0.010000	0.0	17	3	0.850000

From the results, we decide to keep the first one and to improve it. Again we look at the indices of data points that the LF labeled `ABSTAIN` that actually should be labeled as `SU` .

In [35]:

```
buckets = get_label_buckets(L_dev[:, 0], L_dev[:, 1])
df_dev.iloc[buckets[(SU, ABSTAIN)]]
```

Out[35]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Sar
979	980	799	581.67	584.03	B	今天 中午 , 今天 中午 .	4	2.36	
668	669	860	703.48	703.98	A	就是 给 --	3	0.50	
1324	1325	963	196.66	198.23	A1	囊肿 的 很 正常 的 ,	5	1.57	

Our goal is to leave as less as possible abstain votes. So we need other LFs to label them as SU.

To understand the coverage difference between `word_count` and `ywordcount` , let's take a look at 10 data points from the `train` set. Remember: coverage isn't always good. Adding false positives will increase coverage.

In [36]:

```
buckets = get_label_buckets(L_train[:, 0], L_train[:, 1])
df_train.iloc[buckets[(SU, ABSTAIN)]].sample(10, random_state=1)
```

Out[36]:

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	
21135	21149	21150	758	1.96	575.16	NaN	0.64	False	B1	5
7229	7238	7239	1293	2.09	504.29	0.29	0.00	False	B1	5
25887	25904	25905	821	2.49	214.46	0.73	1.48	False	B	2
21516	21530	21531	827	1.51	562.99	0.23	0.00	False	B1	5
10202	10212	10213	703	2.59	617.30	0.29	0.00	False	A	6
16783	16794	16795	763	1.54	218.90	0.00	0.00	False	B	2
22406	22421	22422	721	1.67	594.62	NaN	NaN	False	A	5
20705	20719	20720	111	1.56	403.82	0.25	0.00	False	B	4
24835	24851	24852	1006	2.90	480.75	NaN	NaN	True	B	4
4836	4840	4841	1671	1.73	648.90	NaN	NaN	False	B	6

The results above show some false negatives : ID 4796, 27707, 22952. We will use other indices to increase the accuracy.

I use the most frequent NSU here.

In [37]:

```
from snorkel.labeling import LabelingFunction


def keyword_lookup(x, keywords, label):
    if any(word in x.Text.lower() for word in keywords):
        return label
    return ABSTAIN


def make_keyword_lf(keywords, label=NSU):
    return LabelingFunction(
        name=f"keyword_{keywords[0]}",
        f=keyword_lookup,
        resources=dict(keywords=keywords, label=label),
    )


"""Some final particles are more frequent in NSUs than in SU"""
keyword_fp = make_keyword_lf(keywords=["呀","吧","啦" ])

"""Some feedback words"""
keyword_feedback = make_keyword_lf(keywords=["嗯","啊","啊哈","哎","呃","哈","哦"])

"""wh_words in questions"""
keyword_wh = make_keyword_lf(keywords=["什么","谁","怎么"])

"""modifier words."""
keyword_md = make_keyword_lf(keywords=["应该","大概","可能"])

"""Some words more frequent in SU."""
keyword_su = make_keyword_lf(keywords=["就","说","个","这"], label=SU)
```

In [38]:

```
@labeling_function()
def nsufreqw(x):
    return NSU if x['Text'] in["嗯 ,","嗯 ?","啊 ?","啊哈 .",
                               "哎 ,","哎 .","什么 ?"
                               "对 .","喔 .","哎 ?","呃 .","呃 ,","哈 ?",
                               "哦 .","哦 ," ] else ABSTAIN
```

In [39]:

```
@labeling_function()
def finalparticle(x):
    return NSU if x['Text'] in["呀","吧","啦" ] else ABSTAIN
```

Use the POS-tag information here, in the notebook of *descriptive statistics*, we have compared the data with NSU labels and the whole data set, and there are

some differences.

In [40]:

```
@labeling_function()

def pos1(x):
    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    for tag in tags:
        if tag in ['FW','IJ','OD']:
            return NSU
        else: continue
    return ABSTAIN
```

In [41]:

```
@labeling_function()

def pos2(x):
    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    for tag in tags:
        if tag in ['JJ','LC','CS']:
            return SU
        else: continue
    return ABSTAIN
```

In [42]:

```
## This one is to make sure that cases of word_count =1 not be labeled as SU.

@labeling_function()
def pos3(x):

    bi_tags = nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    bi_tags = list(nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()]))

    if (('AD','PU') in bi_tags) or (('P' ,'PU') in bi_tags)) :
        return NSU
    return ABSTAIN
```

In [43]:

```
@labeling_function()
def pos4(x):

    bi_tags = nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    bi_tags = list(nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()]))

    if (('VA','VA') in bi_tags) or (('NN' ,'VA') in bi_tags)) :
        return NSU
    return ABSTAIN
```

In [44]:

```
HEAVY_TAGS = ['VV','AD','NN','PN']

@labeling_function()


def pos_heavy(x):

    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    heavy = 0
    for tag in tags:
        if tag in HEAVY_TAGS:
            heavy +=1
    if heavy > 1:
        return SU
    elif heavy <= 1 :
        return NSU
    return ABSTAIN
```

In [45]:

```
VERB_RELATED_TAGS = ['AS','VV','VC','M']

@labeling_function()


def pos_verb(x):

    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    heavy = 0
    for tag in tags:
        if tag in VERB_RELATED_TAGS:
            heavy +=1
    if heavy > 1:
        return SU
    elif heavy <= 1 :
        return NSU
    return ABSTAIN
```

In [46]:

```
LIGHT_TAGS = ['AD','P', 'PU']

@labeling_function()

def pos_light(x):

    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    light = 0
    for tag in tags:
        if tag in LIGHT_TAGS:
            light +=1
    if light > 2:
        return SU
    elif light ==2  :
        return NSU
    return ABSTAIN
```

In [47]:

```
@labeling_function()

def has_speaker(x):
    """If several continuous utterances is produced by the same speaker, then it's r
    if x['Same_speaker'] == True:
        return SU
    return ABSTAIN
```

In [48]:

```
L_dev[np.isnan(L_dev)] = 0
Y_dev[np.isnan(Y_dev)] = 0
L_train[np.isnan(L_train)] = 0
```

In [49]:

```
lfs = [pos2,pos_heavy,pos_light,has_speaker]

applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df=df_train)
L_dev = applier.apply(df=df_dev)
```

```
100%|██████████| 31932/31932 [00:02<00:00, 15601.54it/s]
100%|██████████| 900/900 [00:00<00:00, 14848.73it/s]
```

In [50]:

```
LFAnalysis(L_dev, lfs).lf_summary(Y=Y_dev)
```

Out[50]:

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
pos2	0	[0]	0.120000	0.120000	0.008889	105	3	0.972222
pos_heavy	1	[0, 1]	1.000000	0.794444	0.193333	769	131	0.854444
pos_light	2	[0, 1]	0.751111	0.751111	0.170000	519	157	0.767751
has_speaker	3	[0]	0.194444	0.194444	0.067778	139	36	0.794286

3. Combining Labeling Function Outputs with the Label Model

In [51]:

```
lfs = [duration, wordcount,
    yduration,ywordcount,
    english,
    wordcount_disfluency,
    nsufreqw,
    pos2,pos_heavy,pos_light,pos_verb,
    has_speaker,
    keyword_su,
]
```

In [52]:

```
applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df=df_train)
L_dev = applier.apply(df=df_dev)
L_test = applier.apply(df=df_test)
L_valid = applier.apply(df=df_valid)
```

```
100%|████████████████████| 31932/31932 [00:05<00:00, 5478.72it/s]
100%|████████████████████| 900/900 [00:00<00:00, 5305.22it/s]
100%|████████████████████| 300/300 [00:00<00:00, 5264.07it/s]
100%|████████████████████| 301/301 [00:00<00:00, 4939.61it/s]
```

In [53]:

```
LFAnalysis(L=L_dev, lfs=lfs).lf_summary(Y=Y_dev)
```

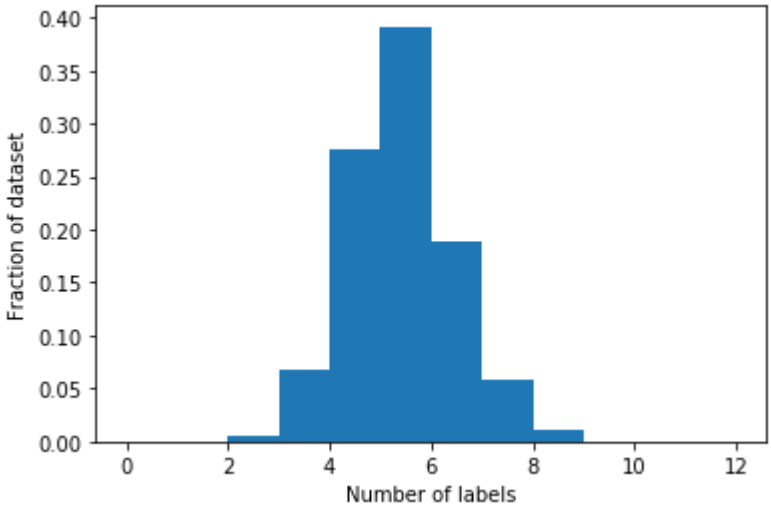
	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Acc.
duration	0	[0]	0.095556	0.095556	0.004444	86	0	1.000000
wordcount	1	[0]	0.335556	0.335556	0.046667	301	1	0.996689
yduration	2	[1]	0.296667	0.296667	0.151111	208	59	0.779026
ywordcount	3	[1]	0.366667	0.366667	0.197778	251	79	0.760606
english	4	[0]	0.016667	0.016667	0.016667	14	1	0.933333
wordcount_disfluency	5	[0]	0.271111	0.271111	0.228889	160	84	0.655738
nsufreqw	6	[1]	0.060000	0.060000	0.024444	52	2	0.962963
pos2	7	[0]	0.120000	0.120000	0.022222	105	3	0.972222
pos_heavy	8	[0, 1]	1.000000	1.000000	0.442222	769	131	0.854444
pos_light	9	[0, 1]	0.751111	0.751111	0.314444	519	157	0.767751
pos_verb	10	[0, 1]	1.000000	1.000000	0.442222	682	218	0.757778
has_speaker	11	[0]	0.194444	0.194444	0.098889	139	36	0.794286

We see that our labeling functions vary in coverage, accuracy, and how much they overlap/conflict with one another. We can view a histogram of how many LF labels the data points in our dev set have to get an idea of our total coverage.

In [54]:

```
def plot_label_frequency(L):
    plt.hist((L != ABSTAIN).sum(axis=1), density=True, bins=range(L.shape[1]))
    plt.xlabel("Number of labels")
    plt.ylabel("Fraction of dataset")
    plt.show()


plot_label_frequency(L_train)
```



We see that over half of our `train` dataset data points have 5 or more labels from LFs. Fortunately, the signal we do have can be used to train a classifier over the comment text directly, allowing it to generalize beyond what we've specified via our LFs.

Our goal is now to convert the labels from our LFs into a single *noise-aware* probabilistic (or confidence-weighted) label per data point. A simple baseline for doing this is to take the majority vote on a per-data point basis: if more LFs voted SPAM than HAM, label it SPAM (and vice versa). We can test this with the `MajorityLabelVoter` baseline model (https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.MajorityLabelVoter.h

In [55]:

```
majority_model = MajorityLabelVoter()
preds_train = majority_model.predict(L=L_train)
```

In [56]:

```
preds_train
```

Out[56]:

```
array([0, 0, 1, ..., 0, 1, 0])
```

However, as we can clearly see by looking the summary statistics of our LFs in the previous section, they are not all equally accurate, and should not be treated identically. In addition to having varied accuracies and coverages, LFs may be correlated, resulting in certain signals being overrepresented in a majority-vote-based model. To handle these issues appropriately, we will instead use a more sophisticated Snorkel `LabelModel` to combine the outputs of the LFs.

This model will ultimately produce a single set of noise-aware training labels, which are probabilistic or confidence-weighted labels. We will then use these labels to train a classifier for our task. For more technical details of this overall approach, see our NeurIPS 2016 (https://arxiv.org/abs/1605.07723) and AAAI 2019 (https://arxiv.org/abs/1810.02840) papers. For more info on the API, see the `LabelModel` documentation (https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.LabelModel.html#sn

Note that no gold labels are used during the training process. The only information we need is the label matrix, which contains the output of the LFs on our training set. The `LabelModel` is able to learn weights for the labeling functions using only the label matrix as input. We also specify the `cardinality`, or number of classes. The `LabelModel` trains much more quickly than typical discriminative models since we only need the label matrix as input.

In [57]:

```
label_model = LabelModel(cardinality=2, verbose=True)
label_model.fit(L_train=L_train, n_epochs=500, lr=0.001, log_freq=100, seed=123)
```

In [58]:

```
majority_acc = majority_model.score(L=L_valid, Y=Y_valid, tie_break_policy="random")
    "accuracy"
]
print(f"{'Majority Vote Accuracy:':<25} {majority_acc * 100:.1f}%")

label_model_acc = label_model.score(L=L_valid, Y=Y_valid, tie_break_policy="random")
    "accuracy"
]
print(f"{'Label Model Accuracy:':<25} {label_model_acc * 100:.1f}%")
```

```
Majority Vote Accuracy:   83.1%
Label Model Accuracy:     80.4%
```

We can also run error analysis after the label model has been trained. For example, let's take a look at 5 random false negatives from the `dev` set, which might inspire some more LFs that vote `NSU`.

In [59]:

```
probs_dev = majority_model.predict_proba(L=L_dev)
preds_dev = probs_dev >= 0.5
buckets = get_label_buckets(Y_dev, preds_dev[:, 1])

df_fn_dev = df_dev[["Text", "Label"]].iloc[buckets[(NSU, SU)]]
df_fn_dev["probability"] = probs_dev[buckets[(NSU, SU)], 1]

df_fn_dev.sample(5, random_state=3)
```

Out[59]:

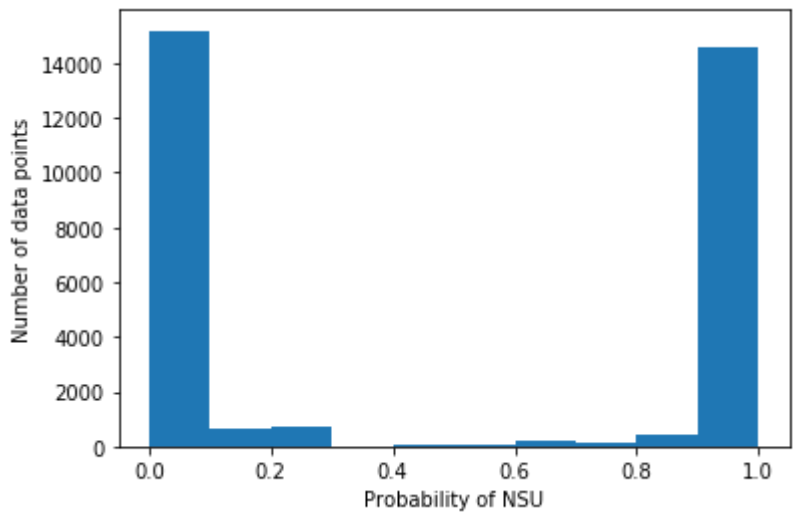
	Text	Label	probability
824	嗯 , 嗯 , 嗯 .	1	0.0
128	%啊% , 应该 不用 吧 .	1	0.0
81	好 爽 , 好 爽 , 好 爽 呃 .	1	0.0
242	对 对 对 对 对 , 就 这 意思 .	1	0.0
202	叫 <English_word_word> 什么 ? <English_for> 什么 ?	1	0.0

Let's briefly confirm that the labels the `LabelModel` produces are probabilistic in nature. The following histogram shows the confidences we have that each data point has the label NSU. The points we are least certain about will have labels close to 0.5.

In [60]:

```
def plot_probabilities_histogram(Y):
    plt.hist(Y, bins=10)
    plt.xlabel("Probability of NSU")
    plt.ylabel("Number of data points")
    plt.show()


probs_train = label_model.predict_proba(L=L_train)
plot_probabilities_histogram(probs_train[:, NSU])
```



Filtering out unlabeled data points

As we saw earlier, some of the data points in our `train` set received no labels from any of our LFs. These data points convey no supervision signal and tend to hurt performance, so we filter them out before training using a built-in utility (https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.filter_unlabeled_dat

In [61]:

```
df_train_filtered, probs_train_filtered = filter_unlabeled_dataframe(
    X=df_train, y=probs_train, L=L_train
)
```

4. Training a Classifier

In this final section of the tutorial, we'll use the noisy training labels we generated in the last section to train a classifier for our task. **The output of the Snorkel `LabelModel` is just a set of labels which can be used with most popular libraries for performing supervised learning, such as TensorFlow, Keras, PyTorch, Scikit-Learn, Ludwig, and XGBoost.** In this tutorial, we demonstrate using classifiers from Keras (https://keras.io) and Scikit-Learn (https://scikit-learn.org).

Featurization

For simplicity and speed, we use a simple "bag of n-grams" feature representation: each data point is represented by a one-hot vector marking which words or 2-word combinations are present in the comment text.

In [62]:

```
vectorizer = CountVectorizer(ngram_range=(1, 2))
X_train = vectorizer.fit_transform(df_train_filtered.Text.tolist())

X_dev = vectorizer.transform(df_dev.Text.tolist())
X_valid = vectorizer.transform(df_valid.Text.tolist())
X_test = vectorizer.transform(df_test.Text.tolist())
```

Keras Classifier with Probabilistic Labels

We'll use Keras, a popular high-level API for building models in TensorFlow, to build a simple logistic regression classifier. We compile it with a `categorical_crossentropy` loss so that it can handle probabilistic labels instead of integer labels. Using a *noise-aware loss* — one that uses probabilistic labels — for our discriminative model lets us take full advantage of the label model's learning procedure (see our NeurIPS 2016 paper (https://arxiv.org/abs/1605.07723)). We use the common settings of an `Adam` optimizer and early stopping (evaluating the model on the validation set after each epoch and reloading the weights from when it achieved the best score). For more information on Keras, see the Keras documentation (https://keras.io/).

In [63]:

```
## This cell makes our Keras results reproducible

import random

import tensorflow as tf


seed = 1
np.random.seed(seed)
random.seed(seed)

session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1, inter_op_parallelism_threads=1
)

from tensorflow.keras import backend as K

tf.set_random_seed(seed)
sess = tf.compat.v1.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [64]:

```
from snorkel.analysis import metric_score
from snorkel.utils import preds_to_probs
from utils import get_keras_logreg, get_keras_early_stopping

# Define a vanilla logistic regression model with Keras
keras_model = get_keras_logreg(input_dim=X_train.shape[1])

keras_model.fit(
    x=X_train,
    y=probs_train_filtered,
    validation_data=(X_valid, preds_to_probs(Y_valid, 2)),
    callbacks=[get_keras_early_stopping()],
    epochs=50,
    verbose=0,
)
```

```
WARNING:tensorflow:From /Users/myday/snorkel-tutorials/.envspam/lib/py
thon3.7/site-packages/tensorflow/python/ops/init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dt
ype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing i
t to the constructor

Restoring model weights from the end of the best epoch.
Epoch 00018: early stopping
```

Out[64]:

```
<tensorflow.python.keras.callbacks.History at 0x139067f98>
```

In [65]:

```
preds_test = keras_model.predict(x=X_test).argmax(axis=1)
test_acc = metric_score(golds=Y_test, preds=preds_test, metric="accuracy")
print(f"Test Accuracy: {test_acc * 100:.1f}%")
```

```
Test Accuracy: 79.3%
```

We can compare this to the score we could have gotten if we had used our small labeled `dev` set directly as training data instead of using it to guide the creation of LFs.

In [66]:

```
keras_dev_model = get_keras_logreg(input_dim=X_train.shape[1], output_dim=1)

keras_dev_model.fit(
    x=X_dev,
    y=Y_dev,
    validation_data=(X_valid, Y_valid),
    callbacks=[get_keras_early_stopping()],
    epochs=50,
    verbose=0,
)
```

```
WARNING:tensorflow:From /Users/myday/snorkel-tutorials/.envspam/lib/py
thon3.7/site-packages/tensorflow/python/ops/nn_impl.py:180: add_dispat
ch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

Restoring model weights from the end of the best epoch.
Epoch 00027: early stopping
```

Out[66]:

```
<tensorflow.python.keras.callbacks.History at 0x14ce2df60>
```

In [67]:

```
preds_test_dev = np.round(keras_dev_model.predict(x=X_test))
test_acc = metric_score(golds=Y_test, preds=preds_test_dev, metric="accuracy")
print(f"Test Accuracy: {test_acc * 100:.1f}%")
```

```
Test Accuracy: 82.3%
```

Scikit-Learn with Rounded Labels

If we want to use a library or model that doesn't accept probabilistic labels, we can replace each label distribution with the label of the class that has the maximum probability. This can easily be done using the `probs_to_preds` helper method (https://snorkel.readthedocs.io/en/master/packages/_autosummary/utils/snorkel.utils.probs_to_preds.html#snorkel.utils.probs_to_preds). It's important to note that this transformation is lossy, as we no longer have values for our confidence in each label.

In [68]:

```
from snorkel.utils import probs_to_preds

preds_train_filtered = probs_to_preds(probs=probs_train_filtered)
```

In [69]:

```
preds_train_filtered
```

Out[69]:

```
array([0, 0, 1, ..., 0, 1, 0])
```

For example, this allows us to use standard models from Scikit-Learn.

In [70]:

```
from sklearn.linear_model import LogisticRegression

sklearn_model = LogisticRegression(C=0.001, solver="liblinear")
sklearn_model.fit(X=X_train, y=preds_train_filtered)
```

Out[70]:

```
LogisticRegression(C=0.001, class_weight=None, dual=False, fit_interce
pt=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001,
verbose=0,
                   warm_start=False)
```

In [71]:

```
Y_train_majority_votes = sklearn_model.fit(X=X_train, y=preds_train_filtered)
```

In [72]:

```
print(f"Test Accuracy: {sklearn_model.score(X=X_test, y=Y_test) * 100:.1f}%")
```

```
Test Accuracy: 76.3%
```

In this notebook, the aim is to build a model of classifying NSUs in our corpus CallHome using Snorkel.

0. Set the environment and import the needed modules

In [1]:

```
#%matplotlib inline

import os

# Make sure we're running from the spam/ directory
if os.path.basename(os.getcwd()) == "snorkel-tutorials":
    os.chdir("visual_relation")

# Turn off TensorFlow logging messages
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# For reproducibility
os.environ["PYTHONHASHSEED"] = "0"
```

In [2]:

```
import os
import re
import string
import glob
import nltk
from collections import defaultdict
from collections import Counter


# Computing / Stats packages
import pandas as pd
import numpy as np

# Classification
from sklearn.feature_extraction.text import CountVectorizer


# Visualisation
import seaborn as sns
import matplotlib.pyplot as plt

# Snorkel
import snorkel
from snorkel.labeling import LabelingFunction
from snorkel.labeling import PandasLFApplier
from snorkel.labeling import LFAnalysis
from snorkel.labeling import MajorityLabelVoter
from snorkel.labeling import LabelModel
from snorkel.labeling import filter_unlabeled_dataframe

from snorkel.analysis import get_label_buckets


DISPLAY_ALL_TEXT = False

pd.set_option("display.max_colwidth", 0 if DISPLAY_ALL_TEXT else 50)
```

1.Read the CallHome corpus and the sample corpus(5% of the data)

In [3]:

```
df_callhome_all = pd.read_csv('Data/calltags_0519.csv')
```

Add a column 'add' to shift back one place with the column 'tagged'

that's to add a column of antecedent content of the column 'tagged', and then to extract the repeated value between the utterance and its antecedent

I wanted to extract only the words, but if I use this below ,by using '_' in the parenthese of split and use the index [0], it only gives me the first repeated words so.

df_callhome_all['word_overlap'] = [set(x[12].split('_')[0]) & set(x[13].split()) for x in df_callhome_all.values]

In [4]:

```
df_v = df_callhome_all[['Tagged']].shift(1)
df_v.columns = ['add']
df_callhome_all = pd.concat([df_callhome_all, df_v], axis=1)
```

In [5]:

```
df_callhome_all['add']= df_callhome_all['add'].fillna("_")
```

In [6]:

```
df_callhome_all['word_overlap'] = [set(x[12].split()) & set(x[13].split()) for x in
```

In [7]:

```
df_callhome_all
```

, ...

...	...	...	...	...	...	...	...	...	...
33456	1067	0.61	449.67	0.16	0.00	False	A	449.06	呃 哟 , 那 就 --
33457	1067	1.33	450.58	0.16	0.42	False	B	449.25	&郭& &中& 打 了 一 个 电话 ,
33458	1067	1.02	451.71	0.11	0.00	False	A	450.69	那 , 估计 号 , (()) [[distortion]]

In [8]:

```
callhome_1500 = pd.read_csv('Data/label_0522.csv')
```

In [9]:

```
callhome_1500
```

Out[9]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Latency
0	1	882	183.47	185.20	A	你很爽哈你,哈?	6	1.73	False	NaN
1	2	882	184.93	185.95	B	要不要跟妈讲话?	5	1.02	False	NaN
2	3	882	186.03	186.45	A	啊?	1	0.42	False	0.08
3	4	882	186.89	187.17	B	啊?	1	0.28	False	0.44

&乖乖& 哦小&生

Also add the same columns for the labeled data

In [10]:

```
df_x = callhome_1500[['Tagged']].shift(1)
df_x.columns = ['add']
callhome_1500 = pd.concat([callhome_1500, df_x], axis=1)
```

In [11]:

```
callhome_1500['add']= callhome_1500['add'].fillna("_")  ## to fill the nan
```

In [12]:

```
callhome_1500['word_overlap'] = [set(x[12].split()) & set(x[13].split()) for x in ca
```

In [13]:

```
callhome_1500
```

	963	412.28	413.67	A	&上海& 白天 九 , 十 点钟 ?	5	1.39	False	NaN	0.90	
	963	414.33	416.95	B	哎 , 就是 你 现在 是 八 点钟 打 唠 ?	9	2.62	False	0.66	0.00	
	963	416.56	417.79	A	哦 , 就 再 过 两 个 小- ,	7	1.23	False	0.66	0.39	

We can subset a class by changing the corresponding label number, to inspect the relative features, such as POS tags, word_count, duration, etc.

In [14]:

```
call_1500_subset = callhome_1500[callhome_1500["Label"] == 8]
```

In [15]:

```
call_1500_subset
```

1213	1214	717	434.29	434.66	A	是 嘛 ?	2	0.37	False	1.05	0.
1222	1223	717	458.17	458.62	A	是吗 ?	1	0.45	False	0.44	0.
1224	1225	717	470.06	470.37	A	是 嘛 ?	2	0.31	True	NaN	N
1250	1251	717	543.01	543.48	A	是 嘛 ?	2	0.47	False	0.70	0.

We load the CallHome dataset and create Pandas DataFrame objects for the development, validation, and test sets.

Each DataFrame consists of the following fields:

- **`Start`** : Start time of the utterance sequence
- **`Text`** : Conversation transcription content
- **`Word_count`** : How many words(punctuations counted separately) in this segment
- **`Duration`** : How many seconds this segment lasts
- **`Overlap`** : The section when more than one speaker speaking at the same time
- **`Latency`** : The pause when the previous speaker has finished but the other speaker hasn't begin
- **`Same_speaker`** : If the two continous sequence is produced by the same speaker
- **`Label`** : Whether the comment is `NSU` (1), `SU(non-nsu)` (0), or `UNKNOWN/ABSTAIN` (-1)

We start by loading our data. We have a manually labeled dataset(call1500) containing the first 1500 lines from the whole CallHome dataframe.

We split the call1500 data into dev, test, valid set by 6:2:2.

In [16]:

```
df_train = df_callhome_all[1500:]

#df_dev = call1500[0:int(len(call1500)*0.8)]
#df_test = call1500[int(len(call1500)*0.8):len(call1500)]



df_dev, df_test, df_valid = np.split(callhome_1500.sample(frac=1), [int(.6*len(callh

print("Train Relationships: ", len(df_train))
print("Dev Relationships: ", len(df_dev))
print("Test Relationships: ", len(df_test))
print("Valid Relationships: ", len(df_valid))

Y_dev = df_dev.Label.values
Y_test = df_test.Label.values
Y_valid = df_valid.Label.values
```

```
Train Relationships:  31932
Dev Relationships:  900
Test Relationships:  300
Valid Relationships:  301
```

2. Writing Labeling Functions

First set the categories.

In [17]:

```
SU = 0 ## The other side of the NSU is SU(Sentential Utterances)
ACKPLAIN = 1
REACK = 2
CHECK = 8
INTERJ = 20
ABSTAIN = -1

print(f"Dev ACKPLAIN frequency: {100 * (df_dev.Label.values == ACKPLAIN).mean():.1f}
print(f"Dev SU frequency: {100 * (df_dev.Label.values == SU).mean():.1f}%")
```

```
Dev ACKPLAIN frequency: 19.3%
Dev SU frequency: 63.7%
```

As we have many tags, there is a great imbalance between the major categories and the minor categories. Successfully classify all the categories may be ideal but a bit unrealistic.

The major categories are :

- 1 : Plain acknowledgement
- 2 : Repeated acknowledgement
- 8 : Check Question
- 20 : Interjection

2.1 Numerical indices used in Labeling Functions

First, we need to sort out the NSU subset and exclude the SU.

In [18]:

```
from snorkel.labeling import labeling_function


@labeling_function()
def duration(x):
    return SU if x['Duration']  > 4  else ABSTAIN


@labeling_function()
def wordcount(x):
    return SU if x['Word_count'] > 6 else ABSTAIN


@labeling_function()
def has_speaker(x):
    """If several continuous utterances is produced by the same speaker, then it's r
    if x['Same_speaker'] == True:
        return SU
    return ABSTAIN
```

Based on the results of the sample data, we impose some limitations of the

Based on the results of the sample data, we impose some limitations of the numerical indices for each class, and use the regex to set some restrictions about the place of the words and the place of the punctuation (sometimes to divide the question sentences from the declarative sentences).

As there are words shared by different categories, our principle is to use the most frequent words for one class of NSU and avoid conflict.

In [19]:

```
@labeling_function()

def lf_ackplain(x):
    if x['Word_count'] < 2:
        if re.search(r"[嗯呃哦嗷哎]", x.Text, flags=re.I) :
            return ACKPLAIN
    return ABSTAIN
```

In [20]:

```
@labeling_function()

def lf_checkq(x):
    if x['Word_count'] < 2:
        if re.search(r"[啊吗嘛]", x.Text, flags=re.I) :
            return CHECK
    return ABSTAIN
```

In [21]:

```
@labeling_function()

def lf_interj(x):
    if x['Word_count'] < 3 :
        if re.search(r"[呀哼哇]", x.Text, flags=re.I) :
            return INTERJ
    return ABSTAIN
```

In [22]:

```
@labeling_function()


def su_overlap(x):
    '''If the common words between the utterance and its antecedent is less than 50%
    words = [utt.split('_')[0] for utt in x['word_overlap']]
    count = 0
    for word in words:
        count +=1
        if count/float(x['Word_count']) < 0.5:
            return SU
    return ABSTAIN
```

In [23]:

```
@labeling_function()


def pos_dis(x):
    '''Normally these POS tags don't appear in NSUs'''
    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    count = 0
    for tag in tags:
        count +=1
    if count < 4:
        if tag in ['VV','VE','AD','NR']:
            return SU
    return ABSTAIN
```

Based on the subset of each class, we apply the features into LFs, for example, the unigram, bigrams of POS tags.

In [24]:

```
@labeling_function()


def ackplain_pos(x):
    tags = [utt.split('_')[1] for utt in x['Tagged'].split()]
    count = 0
    for tag in tags:
        count +=1
    if x['Word_count'] < 2:
        if count < 4:
            if tag in ['NN','AD','VA','PU']:
                return ACKPLAIN
    return ABSTAIN
```

In [25]:

```
@labeling_function()
def ackplain_posbi(x):

    bi_tags = nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    bi_tags = list(nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()]))
    if x['Word_count'] < 2:
        if (('AD','PU') in bi_tags):
            return ACKPLAIN
    return ABSTAIN
```

In [26]:

```
@labeling_function()
def ackplain_postri(x):

    tri_tags = nltk.trigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    tri_tags = list(nltk.trigrams([utt.split('_')[1] for utt in x['Tagged'].split()]
    if x['Word_count'] < 2:
        if ((('NN','VV','PU') in tri_tags) or (('NN','VA','PU') in tri_tags)) :
            return ACKPLAIN
    return ABSTAIN
```

In [27]:

```
@labeling_function()
def check_posbi(x):

    bi_tags = nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    bi_tags = list(nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()]))
    if x['Word_count'] < 2:
        if ((('VC','SP') in bi_tags) or (('DEC','NN') in bi_tags)) :
            return CHECK
    return ABSTAIN
```

In [28]:

```
@labeling_function()
def interj_posbi(x):

    bi_tags = nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()])
    bi_tags = list(nltk.bigrams([utt.split('_')[1] for utt in x['Tagged'].split()]))
    if x['Word_count'] < 3 :
        if (('NN','VA') in bi_tags):
            return INTERJ
    return ABSTAIN
```

In [29]:

```
@labeling_function()


def reack(x):
    '''By limiting the number of words in an utterance, if there is high proportion
    words = [utt.split('_')[0] for utt in x['word_overlap']]
    count = 0
    for word in words:
        count +=1
    if x['Word_count'] < 6:
        if count/float(x['Word_count']) > 0.8:
            return REACK
    return ABSTAIN
```

To apply one or more LFs that we've written to a collection of data points, we use an `LFApplier` (https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.LFApplier.html). Because our data points are represented with a Pandas DataFrame in this tutorial, we use the `PandasLFApplier`

(https://snorkel.readthedocs.io/en/master/packages/_autosummary/labeling/snorkel.labeling.PandasLFApplier.ht
Correspondingly, a single data point `x` that's passed into our LFs will be a Pandas `Series` object (https://pandas.pydata.org/pandas-docs/stable/reference/series.html).

It's important to note that these LFs will work for any object with an attribute named `text`, not just Pandas objects. Snorkel has several other appliers for different data point collection types which you can browse in the API documentation (https://snorkel.readthedocs.io/en/master/packages/labeling.html).

The output of the `apply(...)` method is a ***label matrix***, a fundamental concept in Snorkel. It's a NumPy array `L` with one column for each LF and one row for each data point, where `L[i, j]` is the label that the `j` th labeling function output for the `i` th data point. We'll create one label matrix for the `train` set and one for the `dev` set.

In [30]:

```
lfs = [duration, wordcount,has_speaker,lf_ackplain,lf_checkq,lf_interj,
      ackplain_pos,ackplain_posbi,ackplain_postri,
       check_posbi,interj_posbi,reack, su_overlap, pos_dis]

from snorkel.labeling import LFAnalysis

applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df=df_train)
L_dev = applier.apply(df=df_dev)
```

```
100%|██████████| 31932/31932 [00:14<00:00, 2259.83it/s]
100%|██████████| 900/900 [00:00<00:00, 2154.61it/s]
```

In [31]:

```
LFAnalysis(L=L_train, lfs=lfs).lf_summary()
```

Out[31]:

	j	Polarity	Coverage	Overlaps	Conflicts
duration	0	[0]	0.110610	0.110328	0.000094
wordcount	1	[0]	0.386196	0.333646	0.000000
has_speaker	2	[0]	0.239666	0.207034	0.038895
lf_ackplain	3	[1]	0.120945	0.120945	0.076976
lf_checkq	4	[8]	0.031536	0.031536	0.031536
lf_interj	5	[20]	0.013122	0.011775	0.005794
ackplain_pos	6	[1]	0.195572	0.180728	0.135945
ackplain_posbi	7	[1]	0.054647	0.054647	0.042215
ackplain_postri	8	[1]	0.026556	0.026525	0.012245
check_posbi	9	[8]	0.006796	0.006796	0.006796
interj_posbi	10	[20]	0.013341	0.009238	0.003257
reack	11	[2]	0.127959	0.118283	0.118283
su_overlap	12	[0]	0.480177	0.348052	0.010710
pos_dis	13	[0]	0.048697	0.013936	0.005261

In [32]:

```
## Clear out the NaN in the L_dev and Y_dev
L_dev[np.isnan(L_dev)] = 0
Y_dev[np.isnan(Y_dev)] = 0
```

In [33]:

```
LFAnalysis(L=L_dev, lfs=lfs).lf_summary(Y=Y_dev)
```

Out[33]:

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
duration	0	[0]	0.098889	0.098889	0.000000	88	1	0.988764
wordcount	1	[0]	0.404444	0.361111	0.000000	356	8	0.978022
has_speaker	2	[0]	0.200000	0.171111	0.031111	136	44	0.755556
lf_ackplain	3	[1]	0.102222	0.102222	0.057778	88	4	0.956522
lf_checkq	4	[8]	0.030000	0.030000	0.030000	13	14	0.481481
lf_interj	5	[20]	0.021111	0.017778	0.012222	15	4	0.789474
ackplain_pos	6	[1]	0.168889	0.161111	0.116667	107	45	0.703947
ackplain_posbi	7	[1]	0.053333	0.053333	0.036667	47	1	0.979167
ackplain_postri	8	[1]	0.017778	0.017778	0.005556	13	3	0.812500
check_posbi	9	[8]	0.002222	0.002222	0.002222	1	1	0.500000
interj_posbi	10	[20]	0.014444	0.010000	0.004444	7	6	0.538462
reack	11	[2]	0.118889	0.105556	0.105556	6	101	0.056075
su_overlap	12	[0]	0.567778	0.386667	0.014444	430	81	0.841487
pos_dis	13	[0]	0.024444	0.008889	0.000000	15	7	0.681818

LFs to correct:

From the results above, we see the performance of these LFs could be improved greatly: lf_checkq, check_posbi,interj_posbi, reack

Error analysis

The helper method `get_label_buckets(...)` groups data points by their predicted label and true label. For example, we can find the indices of data points that the LF labeled `ACKPLAIN` that actually belong to class `CHECK`. This may give ideas for where the LF could be made more specific.

The index of the column is the order of the Labeling functions, with python column index starting from 0.

In [69]:

```
from snorkel.analysis import get_label_buckets

buckets = get_label_buckets(Y_dev, L_dev[:, 6])
df_dev.iloc[buckets[(CHECK,ACKPLAIN)]]
```

Out[69]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Late
100	101	882	415.71	416.38	B	啊？	1	0.67	False	0
179	180	1711	266.45	266.87	B	啊？	1	0.42	True	N
230	231	1711	393.76	394.01	A	啊？	1	0.25	False	3
1467	1468	963	374.94	375.30	A	啊？	1	0.36	False	0
1145	1146	717	299.62	300.09	A	好吗？	1	0.47	False	0
976	977	799	579.77	580.15	A	啊？	1	0.38	False	0
1222	1223	717	458.17	458.62	A	是吗？	1	0.45	False	0

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Late
983	984	799	587.68	588.30	B	啊？	1	0.62	False	C
932	933	799	516.60	516.98	A	啊？	1	0.38	False	N
3	4	882	186.89	187.17	B	啊？	1	0.28	False	C
1046	1047	799	698.84	699.32	A	啊？	1	0.48	False	N
127	128	882	473.75	474.06	B1	啊？	1	0.31	False	C
1036	1037	799	682.07	682.36	A	啊？	1	0.29	False	C

Problem to solve

The results above show cases mislabeled as ACKPLAIN but instead should be check question.(According to the lf_check.)

Because a lot of check questions use "吧"(ah), but "啊" (ah) is also commonly used in plain acknowledgement. It's impossible to differentiate them without punctuation.

Another (maybe not so important) problem is in the 'Text' column they are not question yet in the 'Tagged_version'there is question mark, a bit bizarre.

From the False negatives above, maybe we should limit the "word_count" or "duration" more strictly

For example, we can find the indices of data points that the LF labeled `ACKPLAIN` that actually belong to class `INTERJ`. This may give ideas for where the LF could be made more specific.

In [35]:

```
from snorkel.analysis import get_label_buckets

buckets = get_label_buckets(Y_dev, L_dev[:, 5])
df_dev.iloc[buckets[(ACKPLAIN,INTERJ)]]
```

Out[35]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Latency
67	68	882	341.45	341.85	A	呃哼.	2	0.40	True	NaN
1313	1314	963	180.94	181.20	B	对呀,	1	0.26	False	NaN
887	888	799	447.87	448.64	B	哎, 对呀.	2	0.77	False	0.8

Now let's take a look at 10 random `train` set data points where `lf_check` labeled `CHECK` to see if it matches our intuition or if we can identify some false positives.

In [36]:

```
df_train.iloc[L_train[:, 5] == INTERJ].sample(10, random_state=1)
```

36]:

	Unnamed: 0	ID	Conv	Duration	End	Latency	Overlap	Same_speaker	Speaker	Start	Tex
8	30267	30268	1346	0.41	293.08	0.23	0.00	False	A1	292.67	呃 哼
5	14416	14417	916	0.33	595.31	NaN	NaN	True	B	594.98	嗯 哼

Take another example, from the 9th column with the LF "check_posbi" we can find the indices of data points that the LF labeled `SU` that actually belong to class `CHECK`.

In [37]:

```
from snorkel.analysis import get_label_buckets

buckets = get_label_buckets(Y_dev, L_dev[:, 11])
df_dev.iloc[buckets[(SU,REACK)]]
```

Out[37]:

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Late
771	772	799	254.17	256.59	B	不要超过膝盖.嗯.	4	2.42	False	C
1187	1188	717	373.87	374.32	B2	您好.	1	0.45	False	C
1390	1391	963	280.62	281.15	B	人参,	1	0.53	True	N
1154	1155	717	314.30	314.56	B1	记得.	1	0.26	False	C
110	111	882	430.27	430.87	B1	不会啊？	2	0.60	False	C
852	853	799	385.07	385.93	B	都拿到了.	4	0.86	False	C
1333	1334	963	209.37	210.25	A	对,#真#不得了,	5	0.88	False	C
289	290	1711	541.64	542.05	A	不报.	2	0.41	False	C

	ID	Conv	Start	End	Speaker	Text	Word_count	Duration	Same_speaker	Late
701	702	799	159.87	161.12	B	到他家里去啊？	5	1.25	False	0

The examples with "对" should be in ANSWER types and not as INTERJ, so we need to write LFs about the yes-no words features.

Include all the tags to see the performance

In [38]:

```
lfs = [duration, wordcount,has_speaker,lf_ackplain,lf_checkq,lf_interj,
      ackplain_pos,ackplain_posbi,ackplain_postri,
       check_posbi,interj_posbi,reack,su_overlap, pos_dis]

applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df=df_train)
L_dev = applier.apply(df=df_dev)
L_test = applier.apply(df=df_test)
L_valid = applier.apply(df=df_valid)
```

```
100%|██████████| 31932/31932 [00:15<00:00, 2020.96it/s]
100%|██████████| 900/900 [00:00<00:00, 1653.91it/s]
100%|██████████| 300/300 [00:00<00:00, 1339.69it/s]
100%|██████████| 301/301 [00:00<00:00, 1749.53it/s]
```

In [39]:

```
from snorkel.labeling import LFAnalysis

Y_valid = df_valid.Label.values
LFAnalysis(L_valid, lfs).lf_summary(Y_valid)
```

Out[39]:

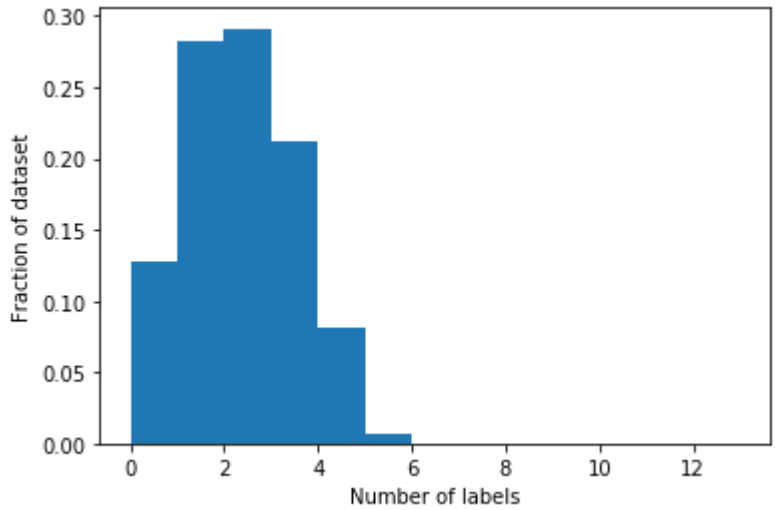
	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
duration	0	[0]	0.112957	0.112957	0.000000	34	0	1.000000
wordcount	1	[0]	0.395349	0.355482	0.000000	116	3	0.974790
has_speaker	2	[0]	0.199336	0.172757	0.043189	40	20	0.666667
lf_ackplain	3	[1]	0.089701	0.089701	0.039867	26	1	0.962963
lf_checkq	4	[8]	0.036545	0.036545	0.036545	6	5	0.545455
lf_interj	5	[20]	0.016611	0.013289	0.009967	4	1	0.800000
ackplain_pos	6	[1]	0.156146	0.146179	0.096346	30	17	0.638298
ackplain_posbi	7	[1]	0.036545	0.036545	0.016611	11	0	1.000000
ackplain_postri	8	[1]	0.026578	0.026578	0.013289	7	1	0.875000
check_posbi	9	[]	0.000000	0.000000	0.000000	0	0	0.000000
interj_posbi	10	[20]	0.009967	0.006645	0.003322	1	2	0.333333
reack	11	[2]	0.162791	0.149502	0.149502	1	48	0.020408
su_overlap	12	[0]	0.538206	0.405316	0.046512	133	29	0.820988
pos_dis	13	[0]	0.029900	0.016611	0.000000	9	0	1.000000

See the average label for each data point, about half of the data have less than 2 labels, about 70% of the data have less than 3 labels.

In [40]:

```
def plot_label_frequency(L):
    plt.hist((L != ABSTAIN).sum(axis=1), density=True, bins=range(L.shape[1]))
    plt.xlabel("Number of labels")
    plt.ylabel("Fraction of dataset")
    plt.show()


plot_label_frequency(L_train)
```



In [41]:

```
majority_model = MajorityLabelVoter(cardinality=21, verbose=True)
preds_train = majority_model.predict(L=L_train)
```

In [42]:

```
preds_train
```

Out[42]:

```
array([ 0,  0, -1, ...,  0,  0,  0])
```

3. Train Label Model

We now train a multi-class `LabelModel` to assign training labels to the unalabeled training set.

In [43]:

```
from snorkel.labeling import LabelModel

label_model = LabelModel(cardinality= 21, verbose=True)
label_model.fit(L_train, seed=123, lr=0.01, log_freq=10, n_epochs=100)
```

In [44]:

```
df_train = df_train.replace(np.nan,'')
df_test = df_test.replace(np.nan,'')
```

In [45]:

```
## Clear out the nan in the L_dev and Y_dev
L_train[np.isnan(L_train)] = 0
Y_test[np.isnan(Y_test)] = 0
```

In [46]:

```
majority_acc = majority_model.score(L=L_test, Y=Y_test, tie_break_policy="random")[
    "accuracy"
]
print(f"{'Majority Vote Accuracy:':<25} {majority_acc * 100:.1f}%")

label_model_acc = label_model.score(L=L_test, Y=Y_test, tie_break_policy="random")[
    "accuracy"
]
print(f"{'Label Model Accuracy:':<25} {label_model_acc * 100:.1f}%")
```

```
Majority Vote Accuracy:   72.0%
Label Model Accuracy:     72.3%
```

In [47]:

```
probs_train = label_model.predict_proba(L=L_train)
```

In [48]:

```
from snorkel.labeling import filter_unlabeled_dataframe

df_train_filtered, probs_train_filtered = filter_unlabeled_dataframe(
    X=df_train, y=probs_train, L=L_train
)
```

In [49]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(ngram_range=(1, 5))
X_train = vectorizer.fit_transform(df_train_filtered.Text.tolist())
X_test = vectorizer.transform(df_test.Text.tolist())
```

We use F1 (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html) Micro average for the multiclass setting, which calculates metrics globally across classes, by counting the total true positives, false negatives and false positives.

In [50]:

```
label_model.score(L_valid, Y_valid, metrics=["f1_micro"])
```

```
WARNING:root:Metrics calculated over data points with non-abstain labe
ls only
```

Out[50]:

```
{'f1_micro': 0.7441860465116278}
```

4. Train a Classifier

You can then use these training labels to train any standard discriminative model, such as an off-the-shelf ResNet (https://github.com/KaimingHe/deep-residual-networks), which should learn to generalize beyond the LF's we've developed!

Scikit-Learn Classifier

In [51]:

```
from snorkel.utils import probs_to_preds

preds_train_filtered = probs_to_preds(probs=probs_train_filtered)
```

In [52]:

```
from sklearn.linear_model import LogisticRegression

sklearn_model = LogisticRegression(C=1e3, solver="liblinear")
sklearn_model.fit(X=X_train, y=preds_train_filtered)
```

```
/Users/chenxinyi/snorkel-tutorials/.envcallhome/lib/python3.7/site-pac
kages/sklearn/linear_model/logistic.py:469: FutureWarning: Default mul
ti_class will be changed to 'auto' in 0.22. Specify the multi_class op
tion to silence this warning.
  "this warning.", FutureWarning)
```

Out[52]:

```
LogisticRegression(C=1000.0, class_weight=None, dual=False, fit_interc
ept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001,
verbose=0,
                   warm_start=False)
```

In [53]:

```
print(f"Test Accuracy: {sklearn_model.score(X=X_test, y=Y_test) * 100:.1f}%")
```

```
Test Accuracy: 74.7%
```