UNIVERSITÉ SORBONNE NOUVELLE PARIS 3

MASTER PLURITAL

WHO'S THE WEREWOLF

Speech comprehension and role prediction in the game of Werewolf

Author:
Manying ZHANG

Supervisor:
Dr. Kim GERDES

September 2018

Abstract

The project is to build a model to predict the identities of the werewolves in the game of The Werewolves of Millers Hollow, aimed for future incorporation in an AI player program. The corpus data comes from the transcription of Chinese TV show Pandakill where 12 players are divided into 4 werewolves, 4 ordinary villagers and 4 divine villagers. The game has several speech and reaction turns until all the werewolves are eliminated, or either all the ordinary villagers or all the divine villagers are eliminated by werewolves. We assume that our AI plays as an ordinary villager to analyze all the speeches and behaviors before its reaction. The speech analysis is composed of 2 steps: the classification of the intent of a single sentence, and the summarizing of sentence meaning and player attitude in a speech turn. The first step is implemented with one of several machine learning and deep learning methods such as Multinomial naive Bayes, logistic regression, Linear support vector machine and Long Short Term Memory model in recurrent neural network; those methods are compared to each other. The second step, along with the behavior analysis part, is built by hand-crafted rules, which in the end gives a most probable Werewolf Team for each game turn.

Acknowledgements

I would first like to thank my thesis advisor M. Kim Gerdes and my teacher M. Sylvain Kahane for their helpful guidance and advice. They gave me complete autonomy in choosing a subject that I love and always showed me an open door when I wanted to discuss the project.

Besides, I would like also to thank my friends and my family, in particular Seginus for his endless loving support and his helpful comments and mathematical suggestions.

Contents

1 Introduction

The Werewolves of Millers Hollow (French: Les Loups-arous de Thiercelieux)[1] is a role-playing game created by French authors Philippe des Pallières and Hervé Marly that can be played with 8 to 47 players. The game is based on the Russian game Mafia[2]. It was nominated for the 2003 Spiel des Jahres award.

There are numerous variants and extensions of the Werewolves game. Since its introduction in China, this game gained a lot of traction there. The unique entertainment and television system in the country has been a fertile ground for many television programs and internet streams where famous stars and web video bloggers play this game, which again, brings it more popularity and variants. In particular, we will refer in our project to the TV show Pandakill[3] to set game rules and retrieve corpus data.

All the variants share the same few basic principles[4]. One of the players serves as the Moderator and the others sit in a circle. The Moderator randomly assign the other players a role, and with each role comes predefined abilities. The roles are kept secret and unchangeable during the whole game. Then the game plays out during cycles of night (when players act) and day (when players discuss and act). There are always two groups of characters : the villagers, some of whom have special abilities (the divine villagers), and the werewolves who kill together a player each night. The goal of the villagers is to find and eliminate all the werewolves by voting for a suspicious player to be executed each day or by using the special abilities of the divine villagers. Similarly, the goal of the werewolves is to eliminate the villagers ; in the "all-to-kill" variant, all villagers have to be killed, while in the "part-to-kill" variant, the wolves need only to kill either all the divine villagers or all the ordinary villagers.

As is the case in the ancestor Mafia game, most of the gameplay revolves around deceit : werewolves need to hide undetected among the players to avoid

[1]`https://en.wikipedia.org/wiki/The_Werewolves_of_Millers_Hollow`
[2]`https://en.wikipedia.org/wiki/Mafia_(party_game)`
[3]`https://zh.wikipedia.org/wiki/Panda_Kill`
[4]For more details, see complete game rules in section 1.3

execution during the daily trials. Villagers also gain some interest in concealing their identity, especially the more powerful divine villagers who are targets of value for werewolves. Also, in the part-to-kill variant, it can also help to waste the werewolves' killing between ordinary and divine villagers.

Our motivation will be to write a chatbot that can serve as an AI player in this game (other than the moderator). Such a program can be decomposed into three parts: speech-to-text, then comprehension of the situation (identities of the other players, how they interact) from the text input, then response generation. In this work, for more simplicity, we would first want it to perform as an extra player out of the original 12 players with a right to observe all information in the game, who will play probably as an ordinary villager in the future and whose position in every speech turn is the last. Additionally, we will focus on the second part, in particular to have a basic understanding of what the players say and guess who the wolves are.

1.1 Project design

The Werewolf's game is a dynamic game with imperfect information: it has several turns and in each turn the players get more information, and they could give different reactions. In order to achieve our goal, we decompose the information into two parts: the speech part and the behavior part. The advantage is that we can process the speech part independently from the game's procedure. Then we add the behavior features to make more logical analysis.

The project is designed in three steps: classification of a single sentence's intent, inference of the relationships between the players, and prediction of players' identities. More precisely, we first classify sentences into basic labels like "attack" (when it accuses a player), "protect", "defense", etc. Then, from this information, we try to describe globally who attacks whom and who protects whom. Finally we guess who the wolves are. The first two steps rely on the speech, and the third uses behavior.

The first step consists in a classical text classification problem. We use and compare several machine learning algorithms to classify the sentence meaning. We have basically two ideas of how to represent the sentence, by a traditional Bag-of-Words theory without considering the context features or by the word embedding with a recurrent neural network which takes context in consideration.

The second step of finding "who attacks whom" asks for the dependency parsing of a sentence. We will use the library HanLP to parse the dependency structure of Chinese sentence. We then reunite these information to make an extraction of the meaning of an entire speech turn of each player.

The last step of the deduction of "who is the werewolf" is based on all the information given by the previous steps, plus the behavior information. Due to the limited quantity of data, we will not use machine learning methods but just a logic induction.

1.2 State of the Art

Since the beginning of modern computing, chatbots have been envisioned as the face of artificial general intelligence, and a functional example was first produced in 1966 with Joseph Weizenbaum's ELIZA [Weizenbaum, 1966]. The famous Turing test proposed in 1950 already aimed at evaluating the performance of such conversational programs. Today, chatbot have found applications well beyond general conversational agents, with some serving as personal assistants in smartphones and home speakers, and others specializing to specific contexts such as customer service.

The dominant paradigm since the beginning of the history of chatbots has been to choose the output (the response) among a finite number of possible response templates : this is the information retrieval paradigm. It ensures that the responses have the highest grammatical quality.

The first implementations of the retrieval based model, starting with ELIZA, are ruled-based systems: the program analyzes the keywords in the user input and chooses the response based on them, using hand-crafted rules. This method has been the dominant one for decades, with a notable example being

Richard Wallace's ALICE[5], a 1995 general conversational chatbot that relies on a database of thousands of rules. As of today, rule-based systems are still competitive enough to be widely used in some industries. For instance, many products of customer service chatbots are designed this way[6].

More recently, advances in computing power and data collection have enabled new techniques that use machine learning to bypass the crafting of complicated rules. Typically, simple models using logistic regression or support vector machines can carry tasks such as sentiment analysis, or classifying between statements, open questions, closed questions, answers, etc. Recent applications include medical interviewer bots designed for diagnosis support [DeVault et al., 2014]. In situations where the space of possible responses is more limited, such as FAQ customer service, classifiers can also be implemented to carry the whole task.

The features used in those models generally involve the words of the input, such as bag-of-words or n-grams. Words can then be encoded discretely, i.e. as vectors whose dimension is the size of the vocabulary.

Recent developments using small neural networks have also allowed for better word embedding, considerably reducing the dimension. Those embedding are typically based on a predictive model for words in a sentence, using simple discrete features as CBOW and skip-grams [Mikolov et al., 2013], and each word is mapped to the corresponding trained parameters of the model. State-of-the-art word embedding word2vec [Mikolov et al., 2013] also mysteriously gets new semantic properties. For example, a reasoning such as "X is to 'woman' what 'king' is to 'man'" can be translated into the vectorial equation "$\text{king} - \text{man} + \text{woman} = X$", and the result X using word2vec is closest to the embedding of "queen"; similarly, $\text{Paris} - \text{France} + \text{Germany} = \text{Berlin}$".

Those word embeddings have notably been leveraged by the latest technique in machine learning, using deep learning to achieve top-grade results. For example, Google rolled out a Smart Reply feature on the world's most popular email platform with great success, using neural networks to predict possible responses

[5]https://en.wikipedia.org/wiki/Artificial_Linguistic_Internet_Computer_Entity
[6]http://southpigalle.io

given an email as input [Kannan et al., 2016].

More impressively, deep learning has allowed to break free from the information retrieval paradigm, building generational models that allow for infinitely more diverse responses while keeping an acceptable grammatical level. The Sequence to Sequence model [Sutskever et al., 2014] is an example of program that builds its own responses word by words (output sequence) given an input sequence, and it can been applied for example to social media manager chatbots [Xu et al., 2017]. Actually, Google's Smart Reply is also based on Sequence to Sequence, although it maps the output sequence to its nearest neighbour among a set of possible responses, to keep the highest grammatical quality provided by the information retrieval paradigm. Generational models can also consider sentences as a whole as opposed to the word-centric Sequence-to-Sequence, drawing on Statistical Machine Translation [Ritter et al., 2011].

In our project, we focus on an approach based on the information retrieval paradigm. Due to the specific nature of the task, namely develop an AI player in a game, it would be best suited to follow the steps of the first implementations of machine learning in chatbot design history (c.f. above). That is to say, we will make use of classical machine learning algorithms (naive Bayes, logistic regression, support vector machine) and deep learning architectures (LSTM) to develop classifiers for text comprehension and to further help analyze the inputs from the other players.

1.3 Werewolf game rules

The game of Werewolf is a role-playing turn-based game. One player, the Moderator, stays out of the game and ensures the good development of the events, in particular by keeping track of the players' status and by telling each player in time what they have to do.

The game unfolds in alternating day turns and night turns starting with the night. During the night, players act using their special abilities. During the day, a player is executed ; before the execution, players talk one by one then a vote is cast to decide which player is to be executed. This will all be explained in more details in section 1.3.b

Although the original game is invented by two French gamers, the game has developed a lot after being introduced in China. Nowadays, the game in France is more like an entertaining party game while in China people tend to play it more seriously. For example, the game in the French official web page[7] allows players to play online, in text, and discussion during the game. But in China, all the discussions are forbidden, players can only speak when it comes to his speech turn. Since our data refers to the professional game of Werewolf in the TV show "Pandakill", we will take their rules and role parameters as ours, instead of the original French version.

1.3.a Roles

All the players except the Moderator are assigned a role (their identity), that is only revealed to the others at the end of the game. We list the different roles with the special abilities that gift them:

Werewolf During each night turn, the werewolves collectively kill one player of their choice. During a day turn, a werewolf can also choose to suicide; this has the effect of eliminating himself and stopping the day turn (so that the next night turn starts immediately).

White wolf The White wolf is the same as a werewolf with the additional ability that when he suicides, he can target one player of his choice who will also immediately die.

Ordinary villager Ordinary villagers do not have any special ability.

Seer During each night turn, the Seer can choose one player and see whether he is a werewolf or not.

Witch The Witch has two bottles of potion, one poison and one antidote. They are to be used during night turns: the poison kills one person of the

[7]`https://www.loups-garous-en-ligne.com`

Witch's choice, and the antidote can revive the player that was killed by the wolves during the same night. Both potions can only be used once, can not use both in the same night, and can only use the antidote on herself during the first night.

Ancient During each night, the Ancient can choose one player; this player will be barred from speaking during the next day, and will only be allowed to communicate with gestures.

Assassin Each night turn, if the Assassin's execution vote the previous day was not in the majority, then he can decide to unilaterally kill the person he voted for to be executed, or not.

Hunter The Hunter has a gun. If during a night turn he is the target of the wolves and the Witch does not save him, or if he is executed during a day, he can shoot one player of his choice, who then dies.

Idiot If the Idiot is executed during a day, he dies but keeps participation in the game as a ghost. During subsequent days, he can interrupt other players' speeches three times per day.

Savior Each night, the Savior can choose a player (including himself) to protect from the wolves (but not from the Witch). This ability can not be used on the same player two consecutive nights.

Werewolves and the White wolf form the Wolf team. The other players form the Villager team. Among the Villagers, we distinguish the ordinary villagers from the others, which we call divine villagers.

1.3.b Mechanics

The two teams (Werewolf and Villager) compete for victory. The villagers win when all the werewolves have been eliminated (i.e. either by the day-time executions, either at night). The wolves' victory conditions depend on the game variant. In the "all-to-kill" variant, they win when all the villagers (divine or ordinary) have been eliminated. In the "part-to-kill" variant, they win when either all the ordinary villagers or all the divine villagers have been eliminated.

As we explained in the introduction of this section, the game is played in alternating night and day turns, starting with the night.

During the night turns, the Moderator calls all players (except the ordinary villagers) one by one to act according to their special abilities. The wolves are called together, so they have the advantage of knowing which players are in their team.

During the day turns, the Moderator starts by announcing the players who have died in the previous night, then the players speak one by one. The purpose of these speeches is to explain guesses about the players' identities; when all players have spoken, a vote is cast to execute one player. Afterwards the executed player can speak once more, then he is eliminated (if this player is the Hunter, his target also gets to speak once more). This closes the day turn, and the following night starts.

There is a special rule for the first day: after the Moderator announces the fatalities (if any) of the first night, the eliminated players can choose to speak before exiting the game, then the round of speeches starts for the remaining players.

Also, at any time during a day turn, a wolf may choose to "suicide by explosion". This has the effect of eliminating himself and of interrupting the turn, so the game directly proceeds to the following night turn (without the remaining speeches and execution).

Strategically, the wolves will try to deceive the villagers during the daytime speeches. Conversely, in part-to-kill games, villagers can also gain interest in deceiving the others, since it makes it harder for the wolves to distinguish ordinary from divine villagers.

The games in our corpus follow three variants:

1. 4 werewolves, 4 ordinary villagers, one Seer, one Witch, one Ancient, one Assassin; part-to-kill.

2. 4 werewolves, 4 ordinary villagers, one Seer, one Witch, one Hunter, one Idiot; part-to-kill.

3. 3 werewolves, 1 White wolf 4 ordinary villagers, one Seer, one Witch, one Hunter, one Savior; part-to-kill.

The Captain There is one more mechanism to the game that we have not explained yet. In order to reduce ties in the execution vote, one player is assigned 1.5 votes. He will be called the Captain.

The Captain is chosen by a vote right before the beginning of the first day, before the Moderator even announces the fatalities of the first night (so all the players are still participating). All the candidates to the position get to speak one by one (in an order randomly decided by the Moderator). Some candidates may quit the election. Afterwards all the non-candidate (excluding candidates that quit) players vote. In the event of a tie (commonly called 'PK' in Chinese), the tied candidates get to speak once more, and all the other players (previous candidates or not) vote. If the vote is still tied, all the players except the remaining tied candidates speak, then those players vote. If there is still a tie, then no Captain is chosen for the game. The first day then proceeds normally (starting with the Moderator review of the first night).

In the event that a wolf "suicides by explosion" the first day before a Captain is elected, then the vote is cast right before the beginning of the second day without the remaining speeches (and if there is a tie, the process continues normally).

When the Captain dies, he can immediately choose a successor (if he died during the night, the succession happens right after the moderator announcement at the start of the following day). He can also choose to "tear his badge", i.e. not designate any successor, in which case there is no Captain anymore.

Strategically, the villagers will want to elect the Seer as Captain, so that in the (likely) event of death, the successor most likely stays in the villager team. Some villagers may be candidate to get some speech time, but they will typically quit before the vote. The wolves will of course also try to get one of them elected.

2 Speech processing

In this part, we will firstly explore the data by introducing the original corpus and explaining how we annotated it, then by giving some statistics of the corpus.

Secondly we will explain how we classified the intent of a single sentence, with the representation of Tf-idf which considers a sentence like a bag-of-word and the representation of word-embedding which takes the context into consideration. For the first part with Tf-idf, we will use classical machine learning methods and in the second part, we will use a recurrent neural network with LSTM.

Finally we will add the syntaxical features to retrieve the subject and object of the sentence, in expectation of making summarising a player's speech turn.

2.1 Data exploration

2.1.a Corpus and annotation

Here we will explain more details about the corpus and the annotation, along with the different processing of the speech and behavior parts.

The corpus is collected from the Chinese TV show Pandakill, so the rules are also based on Pandakill variants. The subtitles are written down and arranged in order. Since the subtitles are what the players speak, we deleted some oral words and made slight modifications to the incorrect grammar.

About how to take down the subtitles of the TV show, first the subtitles are hard written into the video, so there is no subtitle file that could be downloaded. We then tried to contact the producer of the TV program, or some online platform to process the speech to text task like IBM Watson[8], Google Cloud[9], and Chinese specific platform IFlyTek[10], it turned out that the manual notation

[8]https://www.ibm.com/watson/services/speech-to-text/
[9]https://cloud.google.com/speech-to-text/
[10]https://www.xfyun.cn/

is the most convenient and high quality way. We then separate the text into paragraphs of every player's speech turns with their player number. Since the text is from oral language, there might be very long sentences. For the further concern of annotation, we separated the long sentences into short sentences to keep each one a complete and clear sense as possible.

Since the corpus comes from the TV show, all the players play more professionally than in a standard leisure game with a group of friends. They are all clear about the game rules and strategies; they develop the logic well and always have a lot to analyze. However this level of professionality also has drawbacks. The number of players is 12 and the complexity of everyone's speech and logic is improved. For instance, they often use reasoning by contraposition, like "if P2 was really the werewolf, he would not have used this strategy to expose himself".

Due to the complexity of text and the length of a speech, parsing the sentences directly becomes very complicated. To predict a player's character directly by his speech and behavior in several turns is therefore unrealistic, unless we have access to a giant corpus with thousands of recorded and annotated games, in which case the Deep Learning methods might be useful. Since we only recorded 9 games, we have to divide the work into little pieces. A basic idea is to first analyze each sentence or each two sentences separately. Then we analyze the whole speech of a player in one turn. The ideal is to finally make a configuration of the probability distribution of characters for each player. Before getting that far, we first make a configuration of the situation of battle based on the speech and behavior of all players in a turn. Relying on these information and features, we might predict the character of each player.

The original 9 data files use 3 game variants, with 3 files for each variant. They are pure text files and each records a whole game, including the game rule, the announcement of the Moderator and the statements of players. As what we can see below in a little extract of the file, the first paragraph is the announcement of the Moderator:

游戏开始。天黑请闭眼。天亮了，现在进行警长竞选。将要竞选

> 警长的玩家。请举手。1号，2号，5号，9号，12号共五名玩家参与竞选。从2号玩家开始发言。

which means, in English:

> "Night falls. Day comes. Now begins the captain election. The electors please raise your hand. P1, P2, P5, P9, P12 in total 5 players participate in the election. Please give your election speech, start from P2."

Then comes the first statement by a player (here P2), the format is as below:
P2:Name_of_the_player:Statements:Character

For convenience's sake, we replaced the Name_of_the_player by his player number. Also, Chinese numbers appearing before the character 号 were replaced by their Arabic counterpart, for example "二号" becomes "P2". We also annotated the real identity (the role) of the speaking player at the end of the paragraph.

> 2号:2号:预言家先说一下警徽流，警徽流先验7号7号，再验5号5号。呃昨天晚上我查的是12号，12号他是我的金水。过。:狼

In English, this is:

> "P2:P2:I, the Seer, will first give my order of succession as Captain (jinghuiliu) - first to verify (yan) P7, then P5. Last night I checked P12, and he is a good identity (jinshui). Over.:Werewolf"

We will consider the game information as two parts. One is the speech, or the so called statement part, which includes all players' speech turn. Another is the behavior part, which record the votes and the death status, etc. The main part of this paper is the speech text processing and understanding, with in the end some behavior or logic added for the analysis of character. The speech part is retrieved in csv format while the behavior part is noted in json format.

Here we show an extract of the data form in csv format. The column "episode" is the episode number of the game in the TV show Pandakill. The column "timestamp" represents the turn name in the game, for example "captain_election", "night1_lastwords", "day1_speech", "day1_lastwords", etc. More details are explained in section 1.3 in the game's rule. The column "player" represents the player number of the speaking person, while "speech" is what he says. Note that we split a speech turn of one player on smaller sentences, each of them contains a basic meaning unit, which we call "intent", and which is the very last column of the table. The column "character", as already annotated in the original pure text file, is the player's identity.

We translate the first row of the table below:

S1E601 : captain_election : P1: Me, the Seer : Ordinary villager : defense

	episode	timestamp	player	speech	character	intent
0	S1E601	`captain_election`	1号	1号预言家。	民	`defense`
1	S1E601	`captain_election`	1号	昨天晚上查杀是5号牌。	民	`attack`
2	S1E601	`captain_election`	1号	我的警徽流先验9号这张牌，帮大家正一正场上的风气，如果是查杀的话就直接走掉了。	民	`to_check`
3	S1E601	`captain_election`	1号	然后再验呢，再验3号这张牌，3号最近进步太大了，好吧。	民	`to_check`
4	S1E601	`captain_election`	1号	我觉得你们都是一个水平线的玩家，我先验9再验3。	民	`to_check`

Here is the list of in total 8 intents of our labels:

defense claim oneself to be good or answer other people's suspicion

attack accuse other players as being werewolf, or criticize others' behavior as werewolf-like, suspect others of having a bad identity

protect claim others to be good, recognize others' claims as divine, or explain why others are not werewolves

self_attack suicide, abandon the defense, or recognize oneself's mistake

to_check express desire to check some player's identity or persuade the Seer to check him

hang_on not sure of someone's identity and can't distinguish whether he is good or bad

ph summarize the actual situation, analyze a complicated logic or present one's thoughts

none say "over" or make decisions as Captain (like "right hand order"), or talk about irrelevant things or make jokes

We annotate the sense of the sentence just by its superficial sense, without studying the real role of the speaker, neither his relation with the player that he mentions. For exemple, as long as the speaker recognises another as werewolf, whether they are companies or not, he attacks the other one.

As Chinese is a language which does not have spaces between words, we used the library `jieba`[11] to segment the Chinese sentences.

	episode	timestamp	player	speech	char.	intent	segmented
0	S1E601	`captain_election`	1号	1号预言家。	民	`defense`	1-号-预言家
1	S1E601	`captain_election`	1号	昨天晚上查杀是5号牌。	民	`attack`	昨天晚上-查杀-是-5-号牌
2	S1E601	`captain_election`	1号	我的警徽流先验9号这张牌，帮大家正一正场上的风气，如果是查杀的话就直接走掉了。	民	`to_check`	我-的-警徽流-先验-9-号-这-张牌-，-帮-大家-正一正-场上-的-风气-，...
3	S1E601	`captain_election`	1号	然后再验呢，再验3号这张牌，3号最近进步太大了，好吧。	民	`to_check`	然后-再验-呢-，-再验-3-号-这-张牌-，-3-号-最近-进步-...
4	S1E601	`captain_election`	1号	我觉得你们都是一个水平线的玩家，我先验9再验3。	民	`to_check`	我-觉得-你们-都-是-一个-水平线-的-玩家-，-我-...

We can see that in the column "segmented" there are dashes added between words (they are spaces in the original document, we changed them to dashes here and cut some parts for visibility).

2.1.b Statistics

We have in total 4744 sentences split and annotated. As described in the previous part, we have 9 original files, each file records an episode of game in

[11]`https://github.com/fxsjy/jieba`, version number: 0.39

the TV show. Here we can see that among the 9 episodes, the longest has 860 sentences. The day1_speech is the turn with most statements.

	episode	**timestamp**	**player**	**speech**	**char.**	**intent**	**segmented**
count	4744	4744	4744	4744	4744	4744	4744
unique	9	13	12	4659	10	8	4659
top	S3E202	`day1_speech`	5号	过	民	`attack`	过
freq	860	1402	520	35	1639	1604	35

The distribution of each intent label is shown in figure 1.

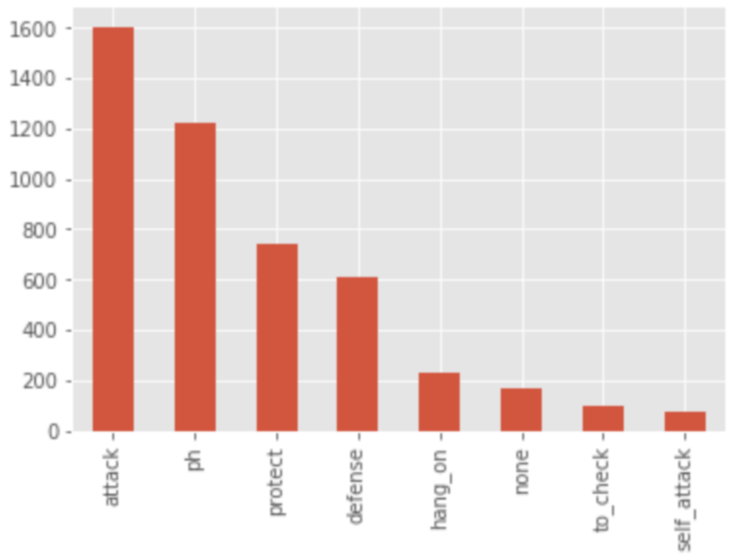


Figure 1: Distribution of the intent labels among the sentences

We can see that the most frequent label is `attack` and the least frequent is `self_attack`.

We then count the length of the segmented sentences. The data are represented in figure 2.

We can see that the longest sentence has 101 words, and shortest just one word. The average sentence length is 20, and 75% sentences have less or equal to 27 words.

In the study of hotwords, we found some special terms used in the game. We list them in the appendix B.

count	4744
mean	20.75
std	11.16
min	1
25%	13
50%	19
75%	27
max	101

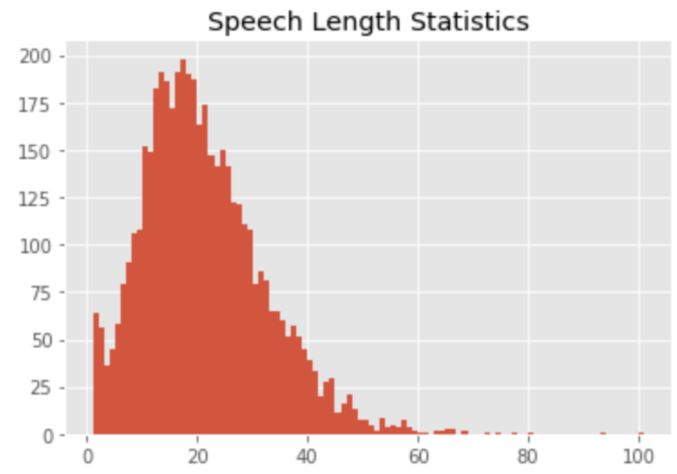


Figure 2: Distribution of the length of the sentences across the corpus

2.2 Intent classification

In this part, we set out to classify the intent of a single sentence (as defined in section 2.1.a) using machine learning methods. We first try classical machine learning algorithms using the Tf-idf representation of Bag-of-Words concept. Then, we will implement a recurrent neural network by representing the text with word embedding in order to take the context into consideration.

We will use only two columns of the annotated corpus: the column "segmented" as texts and the column "intent" as classes. The train and test sets are in proportion 9:1.

2.2.a Without contextual features

In this part, we will introduce the work of classification by some classical machine learning methods. We represent the text by Tf-idf features according to the theory of Bag-of-Words (BOW).

We will mainly use the library Scikit-Learn[12] to do this task, since it has some useful text processing functions like `CountVectorizer` and `TfidfVectorizer`. It also has a complete set of machine learning algorithms among which we will

[12]`scikit-learn.org`, version number: 0.19.1

use naive Bayes, logistic regression and support vector machine (SVM).

2.2.a.i Tf-idf The BOW [McTear et al., 2016] considers a sentence as a bag of words, which means that the order of the words is not taken into account. Hence this thinking simplifies drastically a sentence by removing its contextual and syntaxical features. This makes the text classification tasks using BOW very fast, while it has still been proven to be very useful.

Let's review quickly how a sentence can be represented as a BOW and how can we use it in our classification task.

Suppose that we have a corpus with four sentences: "I love cats", "Cats love fish", "I love cats, cats love fish", "I will eat fish".

We then have a vocabulary of 6 words. We can order these 6 words in a dictionary: `{‘1’: ‘I’, ‘2’: ‘love’, ‘3’: ‘cats’, ‘4’: ‘will’, ‘5’: ‘eat’, ‘6’: ‘fish’}`.

Then we could translate the four sentences to vectors of dimension 6, each coordinate counting the occurences of the corresponding word in the sentence:

"I love cats"	`[1,1,1,0,0,0]`
"Cats love fish"	`[0,1,1,0,0,1]`
"I love cats, cats love fish"	`[1,2,2,0,0,1]`
"I will eat fish"	`[1,0,0,1,1,1]`

This can be done using `CountVectorizer` in Scikit-Learn.

Actually, we can normalize the vectors by dividing the coordinates by the length of the sentence: thus we have replaced the occurrences by the term frequency:

$$\mathrm{TF}(\mathrm{word}, \mathrm{sentence}) = \frac{n(\text{word in sentence})}{n(\text{all words in sentence})}$$

Further, we want to reduce the importance of words that are too common, like "the" or "and", that would give too large components in the vectors. Either we define a stop words list to delete all the common words in the sentences that we don't want to calculate, or we use the Inverse Document Frequency (IDF), which is a function of the number of sentences of the corpus that contain the word:

$$\text{IDF}(\text{word}) = \log \frac{n(\text{all sentences})}{n(\text{sentences having this word})}$$

Or we can use both of them.

The sentences are then represented by vectors where the coordinate corresponding to a word is $\text{TF}(\text{word}, \text{sentence}) \cdot \text{IDF}(\text{word})$[Manning et al., 2008].

The computation of the Tf-idf representations can be done in Scikit-Learn with `TfidfVectorizer`. In fact, this functions allows for tuning of many hyper-parameters, which all give a variant of the Tf-idf representation as explained above. The hyper-parameters are the following:

min_df the words which have frequency lower than the `min_df` threshold will be ignored when building the vocabulary

max_df the words which have frequency higher than the `max_df` threshold will be ignored when building the vocabulary

stop_words the words in the `stop_words` list will be ignored when building the vocabulary

max_features the maximum number of words of the built vocabulary; are kept only the `max_features` words that occur the most frequently across the corpus

ngram_range the range of ngram characters/words when building the vocabulary; for example, if the vaue is `(1,2)`, then only unigrams and bigrams will be considered

norm the method used for normalizing the term frequency

Getting these hyper-parameters right is important for a good Tf-idf vector representation, so that we avoid having too many features or too few features, and that we have the most important features.

We fine-tuned those features with `GridSearchCV`, which was fed the following dictionary, according to their performance on the naive Bayes classification presented in section 2.2.a.ii:

```
{
max_df : (0.1, 0.25, 0.5, 0.75),
min_df : (0.001, 0.0025, 0.005, 0.025),
max_features : (None, 5000, 10000, 50000, 100000),
norm : ('l1','l2',),
stop_words : (stop_ch1,stop_ch2),
ngram_range : ((1, 1), (1, 2), (1,3))
}
```

Since the number of combinations of these parameters is too large, we won't present the results associated with all of them. Here we only give the optimal parameters for the `TfidfVectorizer`: those are `max_df=0.75, min_df=0.005, max_features=None, ngram_range=(1, 3), norm='l2'` and `stop_words` containing also Chinese and Arabic numbers.

Hereafter, we will always use this vectorizer.

2.2.a.ii Naive Bayes naive Bayes is a family of probabilistic algorithms that is widely used in text classification tasks [Schneider, 2005]. They take advantage of Bayes' Theorem in probability theory to predict the class of a text. More precisely, they estimate the probability of each class for a given text, and then output the class with the highest one. The estimation of the probabilities uses Bayes' Theorem, which describes the probability of a feature based on prior knowledge of conditions that might be related to that feature; hence the name of the method.

We will use the multinomial naive Bayes algorithm because we have 8 intents to classify. To better explain what we did and why we chose multinomial naive Bayes to begin, we will briefly review the concept of naive Bayes algorithm along with our data example, for more details, see [McCallum et al., 1998].

We give an example of 4 sentences as below.

	segmented	**intent**
1	I, P1, the Seer.	`defense`
2	I checked P5 last night and he is a villain.	`attack`
3	My order of succession as Captain is first this P9, if he is a villain, I then call for his execution.	`to_check`
4	Then I want to check P3 because he has progressed a lot these days.	`to_check`

For the sentence i, the Bayes classifier will assign the intent $c(i) \in C$ (where C is the set of all intents) with maximal conditional probability:

$$c(i) = \text{argmax}_{c \in C} \mathbb{P}(c \mid i)$$

Bayes' theorem gives

$$\mathbb{P}(c \mid i) = \mathbb{P}(i \mid c)\mathbb{P}(c)/\mathbb{P}(i) \tag{1}$$

The factor $\mathbb{P}(i)$ does not have a real interpretation, but it does not matter since it appears in all the probabilities we want to compare so we can simply ignore it. The factor $\mathbb{P}(c)$ is computed as the frequency of the intent c among the annotated sentences of the training set.

The last factor $\mathbb{P}(i \mid c)$ is trickier. To compute it, we make the naive assumption that the words in the sentence are mutually independent, so that we can write (here for the first sentence of our example)

$$\mathbb{P}(i = 1 \mid c) = \mathbb{P}(\text{"I"} \mid c)\mathbb{P}(\text{"P1"} \mid c)\mathbb{P}(\text{"the"} \mid c)\mathbb{P}(\text{"Seer"} \mid c)$$

We then estimate $\mathbb{P}(\text{word} \mid c)$ as the number of training sentences classified as c where the word occurs, divided by the total number of training sentences classified as c. To avoid having the whole probability driven down to 0 by a single word occurring nowhere in the training sentences labeled c, we apply to every word a Laplace smoothing adding 1 to the occurrences, so that these non-occurring words have a very low non-zero probability.

Thus we can compute, for each sentence i, the $\mathbb{P}(c \mid i)$ for all the classes c using equation 1 and return the class giving the greatest probability.

In fact, many things that can be done to improve this basic model. For example, we can replace the occurrence of words by their Tf-idf features as we introduced before. We can also lemmatize the words to group their different inflection or remove stop words to make the sense more concise. For the Chinese language, the lemmatization might not be so essential as for other languages, because this language doesn't have inflections and the combinations of Chinese characters can produce new meanings while each character individually has its own meaning.

Our implementation of this algorithm uses the module `MultinomialNB` from Scikit-Learn, with the optimal Tf-idf vectorizer found in section 2.2.a.i. There is one hyperparameter to tune, `alpha`, which represents the Laplace smoothing parameter.

When `alpha=0.01`, a best score on the test set is `0.530526315789`. In comparison with the non tuned model, the score is significantly improved. We display here some wrong labels:

index	**speech**	**intent**	**Bayes prediction**
2442	就是我是抿了一下卦象，我是毒了这张10 号牌。	`ph`	`attack`
2365	为什么呢，因为你们12 和2 的发言都很怪。	`attack`	`ph`
1082	我是一个女巫，昨天晚上狼人砍的是3 号，我把3 号捞起来了。	`protect`	`attack`

Translated to English, this is:

index	**speech**	**intent**	**Bayes prediction**
2442	So I made a guess by Pa Kua, I poisoned P10.	`ph`	`attack`
2365	Why? Because you P12 and P2 both have made a strange speech.	`attack`	`ph`
1082	I am a Witch, last knight the werewolves killed P3, I used my power to protect him.	`protect`	`attack`

The sentence 2442 was annotated as `ph` because the speaker expressed was what he thinking about. However the label `attack` given by the classifier would also be acceptable.

The sentence 2365 is certainly an attack on P12 and P2 but is labeled as `ph`. Maybe there is no important keyword to be classified by `attack` and normally `ph` sentences have longer length.

The sentence 1082 has "Witch", "werewolves", "killed", "protect" keywords where "Witch" and "protect" have meaning of `protect`, but "werewolves" and "killed" seem to be more common in `attack`. Perhaps the bayesian probabilities of "werewolves" and "killed" in `attack` is much higher. In addition, the probability of class `attack` is the highest (about $\frac{1}{3}$ across the corpus). We think this is probably the reason for this incorrect label.

So we would like to find something else than statistical probability or the so-called "generative classifiers" like naive Bayes algorithm, we then move to "discriminative classifiers" such as logistic regression to look for new possibilities [Ng and Jordan, 2002].

2.2.a.iii Logistic regression Logistic Regression is a very classical algorithm for classification tasks. We will first briefly explain the principle of logistic regression [Fan et al., 2008], then we will use Tf-idf representation to classify the text intent by logistic regression classifier in Scikit-Learn.

As in any regression problem, the goal of logistic regression is to fit labeled data (x_i, y_i) with a function $y = f(x)$, by choosing the f from a predefined class of functions $\{f_w\}$. For example, linear regression will fit vectorial data with a linear function. Here, we have a classification problem, so the x_i are vectors in a space $\mathbb{R}^d$ but y_i are in the discrete set $\{0, 1\}$, and the model function f gives to the vector x the label closest to $f(x)$. Given this, linear functions would not give a good fit. Instead, we want functions that are close to 0 in a half-space and close to 1 in the other half-space, as in figure 3.

This can be achieved using the sigmoid function $S(t) = \frac{1}{1+e^{-t}}$. The class of functions we consider is then

$$f_{w,b}(x) = S(w \cdot x + b) = \frac{1}{1 + e^{-w \cdot x - b}}$$

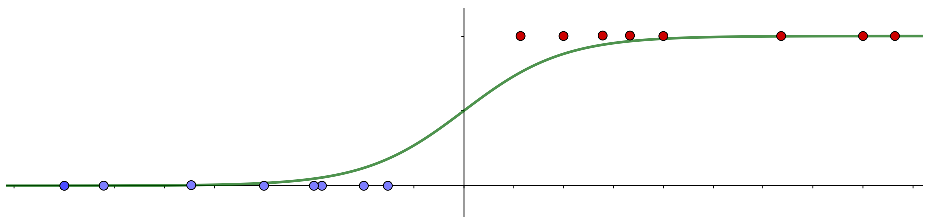


Figure 3: The sigmoid function is suited for binary classification. Created with GeoGebra 6.

where w is a vector of the same dimension as x and b is a scalar. Note that this is classification by hyperplane: the parameters of the model to compute decide the position of the separating hyperplane.

Since the sigmoid function outputs a value in the interval $[0, 1]$, we can interpret it as a probability: we will say that the model of parameters (w, b) assigns to the input x the label 1 with probability $f_{w,b}(x)$ and 0 with probability $1-f_{w,b}(x)$. Then given an observed data (x, y) (with $y \in \{0, 1\}$), the data corresponds to the model prediction with probability $\mathbb{P}(y \mid x, w, b) = f_{w,b}(x)^y(1 - f_{w,b}(x))^{1-y}$. Assuming independence between observation, the likelihood that the model will output all of our data (x_i, y_i) becomes

$$\begin{aligned} L(w,b) &= \prod_i \mathbb{P}(y_i \mid x_i, w, b) = \prod f_{w,b}(x_i)^{y_i}(1 - f_{w,b}(x_i))^{1-y_i} \\ &= \prod_i \left(\frac{1}{1+e^{-w\cdot x_i - b}}\right)^{y_i} \left(\frac{1}{1+e^{w\cdot x_i + b}}\right)^{1-y_i} \end{aligned}$$

and our objective is to find the parameters (w, b) that maximize this likelihood, or equivalently that minimize the inverse loglikelihood

$$-\log(L(w,b)) = \sum_i y_i \log(1 + e^{-w\cdot x_i - b}) + (1 - y_i)\log(1 + e^{w\cdot x_i + b})$$

. This is also called the *cross entropy error*. Generally, people also add to this error a *regularization* term in $\frac{1}{2C}(\|w\|^2+b^2)$ where C is a scalar hyperparameter[Ng and Jordan, 2002], aimed at preventing the parameters (w, b) from becoming too wild (to prevent overfitting). The more C grows, the less this has an influence on the result.

The minimization problem is typically addressed by a gradient descent technique: starting from a point (w, b) in the parameter space, the gradient of the cross entropy error in this point is computed, then the point is updated by a small increment in the direction opposite to the gradient. After enough iterations, a local minimum should be reached.

In our example, we have 8 intents to classify, so it is a multiclass classification task and not a binary one. There are several approaches to this problem. One is to pick out a class k and produce a binary logistic regression model f_{w_k,b_k} that should output 1 for data of class k and 0 for data of class different than k. This can be done for each class, so that we get one model f_{w_k,b_k} per class k. To merge them together, we simply decide that an input vector is mapped to the class of highest probability, that is to k such that $f_{w_k,b_k}(x)$ is maximal. This method is called one-vs-rest logistic regression (OVR), and is naturally best suited for problems where each classification problem is independent.

Another common approach is called multinomial logistic regression. Its idea is to model the probabilities of each class k (taken from a set $\{1, \dots, K\}$ as proportional to $e^{w_k \cdot x + b_k}$, where the w_k and b_k are the parameters of the model. Therefore, according to the model, the probability that the class of x is j is $\frac{e^{w_k \cdot x + b_k}}{\sum_{l=1}^{K} e^{w_l \cdot x + b_l}}$, so the likelihood to maximize is

$$L(w_1, b_1, \dots, w_K, b_K) = \prod_i \frac{\sum_{k=1}^{K} \delta(y_i = k) e^{w_k \cdot x + b_k}}{\sum_{l=1}^{K} e^{w_l \cdot x + b_l}}$$

with also possibly a regularization term $\frac{1}{2C}(\sum_j \|w_j\|^2 + b_j^2)$.

For our work, we use the function `LogisticRegression` from Scikit-learn, and we tune the following parameters using `GridSearchCV` :

multi_class the approach to multi-class logistic regression, that is `'ovr'` or `'multinomial'`

solver the algorithm to use in the optimization problem when training the parameters. Note that only `'newton-cg'`, `'sag'`, `'saga'` and `'lbfgs'` can be chosen for multinomial loss, and `'liblinear'` is for OVR

C inverse of regularization strength, this is the scalar that appears in the regularization term.

According to the Grid Search result, the optimal hyper-parameters are: `multi_class='ovr', solver='liblinear', C=10`. It gives a best score on test set of `0.532631578947`.

There is also an interesting phenomene here: the three sentences that we analyzed as examples of misclassification by the naive Bayesian algorithm are again incorrectly labeled.

index	**speech**	**intent**	**LR prediction**
2442	So I made a guess by Pa Kua, I poisoned P10.	`ph`	`defense`
2365	Why? Because you P12 and P2 both have made a strange speech.	`attack`	`ph`
1082	I am a Witch, last knight the werewolves killed P3, I used my power to protect him.	`protect`	`attack`

However this time sentence 2442 is classified as `defense`. It might be acceptable because "I poisoned P10" could be seen as equivalent to a personal statement of "I am a Witch" since only the Witch has poison, and the declaration of one's own good identity could be labeled as `defense`. Maybe this sentence is itself too ambiguous.

According to Scikit-Learn's official documentation[13], the support vector machine algorithms are supposed to have good performance in text classification tasks. Since the results of naive Bayes and logistic regression are very close, we will try SVM algorithm in the next part.

2.2.a.iv Support vector machine Another variant of classification by hyperplane is the support vector machine (SVM)[Ng, 2018]. Like logistic regression, an SVM will solve a binary classification problem by drawing a hyperplane that best separates the two classes. The major difference is that the SVM draws the hyperplane so that it is the farthest possible from any data point, ie there is

[13] `http://scikit-learn.org/stable/modules/svm.html`

a widest possible "no man's land" along the hyperplane and without any data point inside, as in figure 4.

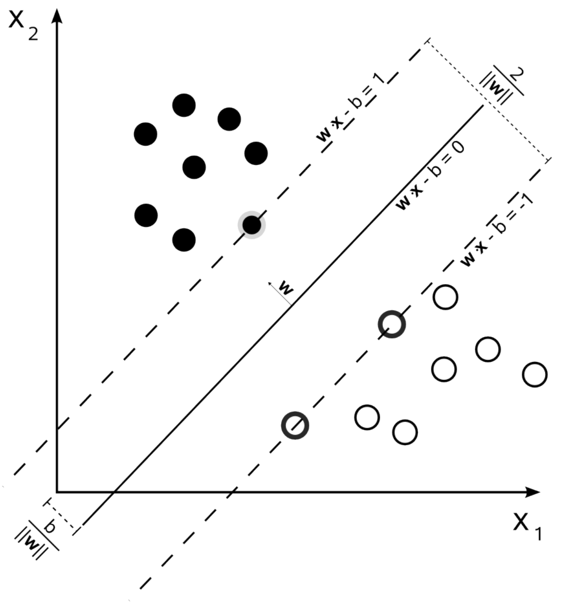


Figure 4: The SVM tries to separate the to classes with a largest possible strip. Source: `https://commons.wikimedia.org/w/index.php?curid=3566688`

The result is that only the data points close to the hyperplane (i.e. at the "boundary" of their class) are taken into account; those data points are the so-called support vectors. Hence the points far from the "boundary" (i.e. well "inside" their class) do not influence the choice of the hyperplane, while in a logistic regression all the data points would have some influence.

More precisely, given data points $x_i \in \mathbb{R}^d$ with labels $y_i \in \{-1; 1\}$, we want to find an hyperplane $w \cdot x + b$ (with normal vector w) such that all the data points are correctly classified, i.e. $\mathrm{sign}(w \cdot x_i + b) = y_i$, and where the minimal distance $\frac{1}{\|w\|} y_i(w \cdot x_i + b)$ of a data point to the hyperplane is maximized. We can rescale w and b such that the minimum of the $y_i(w \cdot x_i + b)$ is 1, then the distance to maximize is $\frac{1}{\|w\|}$. This corresponds then to the minimization problem

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|w\|^2 \\ &\text{subject to} && y_i(w \cdot x_i + b) \geq 1 \end{aligned}$$

However, the problem is not always solvable : if the two classes are somewhat mixed together, there may not exist a hyperplane that separates them perfectly. Therefore we have to allow for some error (data points x_i wandering at some distance $-\frac{1}{\|w\|}y_i(w \cdot x_i + b)$ at the wrong side of the hyperplane), and penalize them using a squared hinge function

$$\text{sqhinge}(\xi) = \begin{cases} 0 & \text{if } \xi > 0 \\ \xi^2 & \text{if } \xi \leq 0 \end{cases}$$

Our problem becomes

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|w\|^2 + C\textstyle\sum_i \xi_i^2 \\ &\text{subject to} && y_i(w \cdot x_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \end{aligned}$$

where $C > 0$ is a regularizing parameter. This formulation allows the problem to be solved with Lagrange duality.

The choice of C is essential as we can see in the figure 5.

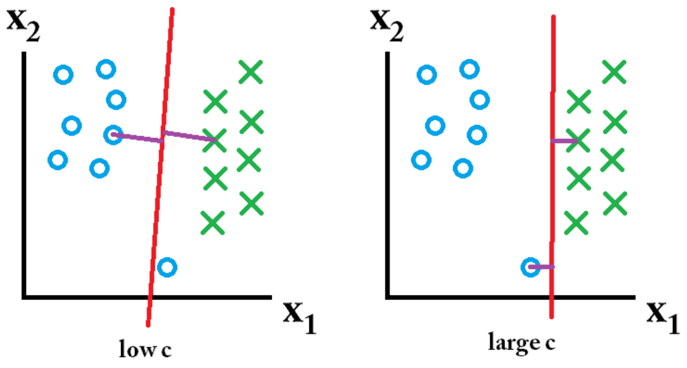


Figure 5: Influence of the regularization parameter C. Source: `https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel/159051`

When C is high, it becomes very important to classify as many data points correctly as possible, at the cost of a small distance between points and the hyperplane. Conversely when C is low, classification errors do not matter much if it can allow for a wider gap along the hyperplane (the error stemming from a few points in the gap is driven very low by C). To summarize heuristically, setting a lower C avoids overfitting at the cost of fidelity to the data points.

In our work, the SVM is implemented with `LinearSVC` from sklearn (hence the default squared hinge error instead of the more common hinge), with the default one-vs-rest strategy for multiclass classification. The hyper-parameter C is tuned with `GridSearchCV`.

We found that C=0.3 gives a best score on test set of 0.547368421053. This is the highest accuracy compared to naive Bayes and logistic regression, but even then we found the three example sentences to be still incorrectly labeled:

index	**speech**	**intent**	**SVM prediction**
2442	So I made a guess by Pa Kua, I poisoned P10.	`ph`	`defense`
2365	Why? Because you P12 and P2 both have made a strange speech.	`attack`	`ph`
1082	I am a Witch, last knight the werewolves killed P3, I used my power to protect him.	`protect`	`attack`

The misclassifications here are identical to those of logistic regression (in particular for sentence 2442)

Here are some sentences that are incorrectly classified by logistic regression, but are correctly labeled by SVM:

index	**speech**	**intent / SVM pred.**	**LR prediction**
2639	他的视角，还有他的心态，一定是一张预言家牌。	`protect`	`attack`
3670	不要说什么通过12 号的发言，5 号的发言来判断我的身份。	`defense`	`attack`
2218	你验他的理由太牵强了，他验我，我能接受。	`attack`	`ph`

In English:

index	**speech**	**intent / SVM pred.**	**LR prediction**
2639	His perspective of view, plus his attitude, he must be a Seer	`protect`	`attack`
3670	Don't say that you judge my identity according to the speech of P12 or the one of P5	`defense`	`attack`
2218	The reason that you verified him is completely out of the point, I can accept that he verifies me though	`attack`	`ph`

The SVM seems to perform better than logistic regression on the unequivocal sentences like 2639, though not specifically better on the ambiguous one.

2.2.b With contextual features

The best results of the above algorithms are around 53%, while normally they would do a good job in text classification tasks. We suppose that it is due to the lack of consideration of context features.

Many text classification tasks aim to classify the topic of the text or the sentiment of a review. In this case, using word Tf-idf features may represent the sentence appropriately enough because the topic or the sentiment is highly related to the words that we use in the sentence. The order of the word or the so called context features, might not be so determinative to the task.

However in our case, when we sorted the most correlated words for each intent, we found that many words overlap across intents. Therefore it might be a good idea to take context features to consideration.

To this end, we can try to consider a sentence as a sequence of words instead of a bag of words. This representation is especially suited to a recurrent neural network, as it processes the input sentence word by word and reuses former information recursively during the propagation, thus keeping some measure of the contextual information.

The words of the sentences, as parts of the input in the neural network, need to be represented as vectors. We could simply take the dimension to be the size of the vocabulary and have a word be a vector of zeros and a one, however this makes the dimension unnecessarily big. Instead, we can use a *word embedding*, that is a way to represent words as vectors in a low-dimensional space while still retaining the semantic relationships between them through the geometry of the space.

2.2.b.i word embedding To talk about word embedding, we would first like to begin with the concept of a simple Neural Network [Hansen and Salamon, 1990]. A neural network is made of neurons which are connected through computations where informations flows (weights for out computational model), and when we train a neural network we want the neurons to fire whenever they learn specific patterns from the data. The fire rate is modeled using an activation function.

Neurons in the networks are arranged in layers. It will have an input layer h_0 which just represents the input information as a vector x and passes it to the next hidden layer h_1 "almost" linearly: the layer h_1 gets the result of the product $W_1 h_0$ where W_1 is a matrix. An activation function g_1 is then used on this result, so that ultimately the information got by h_1 is non linear in the input. This is to prevent the whole network from being linear as composition of linear steps (the hidden layers would then be pointless). The activation function could be the sigmoid we introduced before, it could also be other functions. The information of h_1 is then passed to the layer h_2 in a similar fashion and so on. At the end of the hidden layers we have the output layer. In this layer, an activation function like 'softmax' (similar to sigmoid) is used to get the probability of each class and then chose the one which has highest probability.

As an example, consider the following word prediction problem in [Bengio et al., 2003]. Suppose we have a text corpus with a vocabulary of size V. Our task is to predict a word w_t in a sentence given the previous and following words w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2}, etc. This is a classification task: w_t is the class that we would like to classify the given words into.

We will achieve this task by training our data through an one-layer neural network, represented in figure 6. First, we input the context words using a one-hot encoding. We then set the hidden layer to be a matrix of dimension $n \times m$, where the horizontal length is an integer m and the vertical length is the number

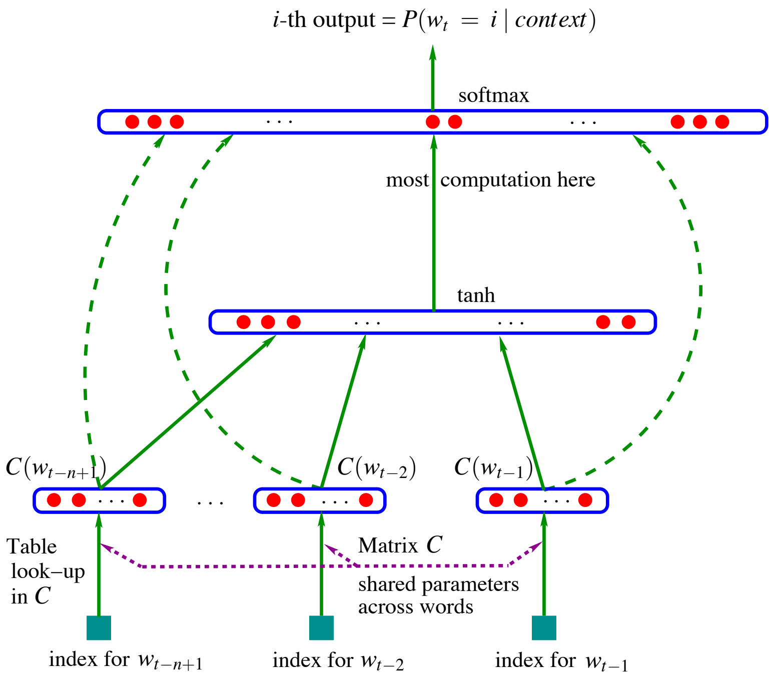


Figure 6: Prediction of a word based on the context. For our explanation we can ignore the second layer with tanh. Source: [Bengio et al., 2003]

n of input context words. It is computed from the input layer using a matrix C of dimension $m \times V$: the i-th lign of the hidden layer will be is $C(w_i) = Cx_i$ where x_i is the i-th input one-hot vector (corresponding to w_i). Note that for this first layer we do not use an activation function, this part is linear. From this we compute the output layer $(y_1, ..., y_V)$ of size V, and we apply a softmax function to assign to the j-th word in our vocabulary the probability $\frac{e^{y_i}}{\sum_{j=1}^{V} e^{y_j}}$ that this word is the word w_t to predict.

After this neural network is trained, we get in particular the matrix C, of dimension $m \times V$ of trained weights for the first layer. This gives an embedding of our vocabulary into the vector space $\mathbb{R}^m$. Such embeddings can retain many interesting semantic properties, as explained in the introduction section 1.2.

Nowadays, people publish many works of pre-trained word embeddings for many languages based on different corpora, or with different models and vector dimension. We chose the Chinese word embedding [14] based on Chinese Quora with the state-of-the-art model `word2vec`. Compared to other corpus, we think that this corpus is a good compromise, having not too many oral characters nor being too serious like People's Daily.

2.2.b.ii Recurrent neural networks In the previous part, we introduced a simple neural network architecture. In fact, the neural network family is huge, with feedforward neural networks like perceptrons, back propagation networks and convolutional neural networks, as well as feedback neural networks like recurrent neural networks (RNN).

The RNN are proved to have better performance in NLP sequential tasks thanks to their recurrent property [LeCun et al., 2015]. The basic concept of RNN uses the time component, and is shown in figure 7.

The basic graph of the network is represented by the one on the left, and its unfolding in time is the one on the right: it is comprised of one hidden layer s which is recursively updated (so it is in fact multiple hidden layers s_0, s_1, etc). The network is fed the input x by successive bits x_t, and the layer s is computed

[14] `https://github.com/Embedding/Chinese-Word-Vectors`

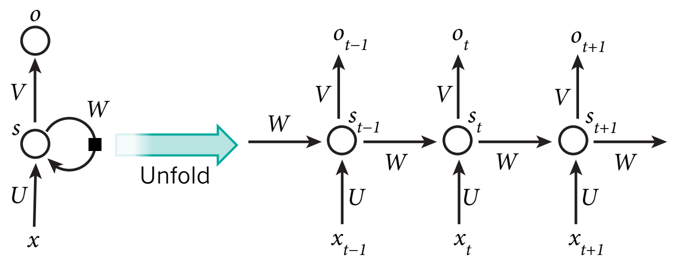


Figure 7: An RNN is represented by the graph on the left; unfolding along the time component gives the graph on the right. Source: [LeCun et al., 2015]

from the last input x_t using weights U and from the previous iteration s_{t-1} of itself using weights W, with an activation function that could be softmax, tanh or rectified linear unit (ReLU). At each iteration, an output o_t can be produced using weights V. Since the matrices U, V and W are the same at each iteration, we can say that the network keeps previous information, as if having memory.

For general NLP applications, we can see x as a sentence, and x_t is the embedding of the t-th word in x. In this setting, the property of retaining memory can be translated by saying that the contextual features are taken into consideration.

In the real-life implementation, people often use two variants of RNN: LSTM (Long Short Term Memory) and GRU (Gated Recurrent Unit) [Chung et al., 2014]. We will use the LSTM variant, that we first briefly explain.

The invention of LSTM addresses the problem of vanishing and exploding gradient [Bengio et al., 1993]. Basically, this problem arises in the original RNN models, where an earlier input will have less influence compared to a closer input. So the output is often taking more information from new input, that's to say the RNN cannot have a long term memory.

LSTM then introduces a concept called constant error carrousel (CEC), this is to make the ideal activation function a linear function. So letting W be the identity and the activation be $f(x) = x$ will keep the error constant. We then have $s_t = s_{t-1} + Ux_t$. But then there will be the problem of updating the U and V confliction. To solve this problem, people artificially create two gates: an input gate g_{in} and an output gate g_{out}. Then $s_t = s_{t-1} + g_{\text{in}} U x_t$, and $o_t = g_{\text{out}} s_t$. Furthermore, people have recently made many improvements [Chung et al., 2014] like adding a forget gate to memorize and forget the CEC in self-connection.

In this project, we used the LSTM module in Keras[15]. We used an LSTM with self-trained word embedding, another with pre-trained Chinese word embedding and set trainable or not. There are many hyper-parameters that should be taken into consideration:

batch_size the number of samples in a time when training, called batching epochs: the epochs of training

embedding_dim the dimension of embedding vector, that we set to be the same as the pre-trained word embeddings (300)

activation the activation function at the end of the hidden layer

dropout regularization method where inputs to LSTM units are probabilistically excluded from activation and weight updates while training a network

recurrent_dropout dropout for recurrent connections

optimizer the optimization methods: could be SGD, Adagrad, etc.

lr the learning rate in optimizer, a big `lr` makes gradient descent go at a big pace

Through a grid search for the hyper-parameters, we obtained that the optimal parameters for the LSTM with self-trained word embedding are:
`batch_size=100, epochs=20, embedding_dim=300, activation='sigmoid', dropout=0.4, recurrent_dropout=0.4, optimizer='adam', lr=0.01`.

As we introduced in the corpus statistics section 2.1.b, we find that the average sentence length is 20, and 75% of sentences have fewer or equal to 27

[15]`https://keras.io`, version number: 2.0.6

words. We present the test accuracy with the hyper-parameters above and with the input sentence length padded into 15, 20, 30, 40, 50 as below.

	self-trained	**pretrained**	**trainable**
10	0.410526316417	0.389473690798	0.412631575999
20	0.446315785772	0.471578950945	0.450526320621
30	0.461052637351	0.48421051314	0.458947369927
40	0.465263153377	0.498947369425	0.46736842237
50	0.448421040648	0.442105260335	0.433684207891

From the table we can see that, with the same hyper-parameters, the pretrained word embedding gives best performance to this task. Compared to other input length, padding each sentence to 40 words generally gives the best results.

2.2.c Comparing the results

The benchmark performance of the classifiers we implemented in the previous parts is as below:

support vector machine + Tf-idf	0.547368421053
Multinomial Logistic Regression + Tf-idf	0.532631578947
naive Bayes + Tf-idf	0.530526315789
LSTM + pretrained word embedding	0.498947369425
LSTM + pretrained word embedding + continuous training	0.46736842237
LSTM + self-trained word embedding	0.465263153377

The support vector machine with Tf-idf gives the best results, with a score of 55%. In the following parts of our project, we will then keep it as our intent classifier.

If we use a random classifier (i.e. with output independent of the input), the optimal way is to always predict the most frequent intent `attack` (which occurs for 33.8% of the corpus sentences), and the random baseline accuracy would be 33.8%. Thus our trained classifiers are a net improvement from the baseline. Also, previous analysis pointed out that many misclassifications stem from data that was annotated ambiguously (where another intent could also have been annotated), so our best score of 55% is in fact a lower bound of the "real" effectiveness.

One disappointment however is the LSTM: compared to the other algorithms along with Tf-idf representation, the neural network is not as good as we expected. It seems that even with the advantage taking into consideration the context features, the classifier does not perform well. There are still many things to do which might bring some improvements to the RNN, such as add more LSTM layers, collect more data or add syntactical features. In this case, the hyper-parameters should also be re-tuned. Given more computation time, we could follow this lead to probably get interesting results.

In the next parts, we will focus on text meaning understanding in favor of the syntactical features retrieved by HanLP then summarize all input information for further use.

2.3 From classification to comprehension

In the previous parts, we used several algorithms to classify the main intent of a sentence, but we still do not know what the meaning of the sentence is. If a sentence expresses intention of attacking someone, we would like to know which person is the target. Besides, since the classifiers always give a score of about 50%, we will not rely only on the intents that are labeled. Instead, we will take syntaxical considerations in order to understand the meaning of a sentence.

At first, we will introduce a Chinese language processing library HanLP [16] to parse the sentence. In the next step, we will explain how we use the dependency information, along with the intent label to extract the meaning of the sentence.

Notice that, from this step, the information is no longer totally independent to each other. The sentences of a player in a speech turn will be finally summarized and then based on that, the attitude of all players in a speech turn will also be summarized.

Hence we cannot process them in disorder such as what we do in the clas-

[16] `https://github.com/hankcs/HanLP`

sification task. We should choose a complete game to process its turns. The S1E101 game is chosen to be the test data. By concern of not leaking the test data in the train data, we use other episodes' data as train set and we use only the hyper-parameters that are fine tuned in section 2.2 to retrain a classifier of intent, then apply on the test file. The classifier `LinearSVC` is chosen because of its good and fast performance.

2.3.a Who attacks whom

In this part, we will first extract from each sentence its basic "subject verb object" structure. We will then, secondly, use this structure, combined with the classifier produced in the previous section, to extract some explicit information conveyed by the sentence, such as "who attacks whom", "who protects whom", "who defends himself against whom", etc.

There are basically two ideas to realize this second extraction: either we use the sentence intent labeled with the classifiers in 2.b as the verb "attack" or "protect" etc, then use the subject and object retrieved as "who" and "whom"; or we relook the verb retrieved to see whether it corresponds to the labeled intent. Indeed in some cases, players use specific verbs (cf. glossary B) that give an explicit indication of their intent. These indicated intents might be different to the labeled ones. As we have said before, the accuracy of our classifiers is always around 50%, we would better not to only rely on the classifiers to take labeled intent as meaning verbs, so these two ideas will both be used in the section.

In this part, we will use the library HanLP[17] to parse the dependency of a sentence. HanLP is a powerful open source tool that focuses on Chinese processing in many basic tasks like Chinese word segmentation, POS tag, named entity recognition, dependency parsing, etc, as well as more complicated tasks like keyword extraction, automatic summary or text classification.

The reason that we chose HanLP is its convenience and rich abilities. We

[17]`https://github.com/hankcs/HanLP`, version number: 0.1.41

had used jieba[18] for Chinese word segmentation for its fast and satisfying performance, and it allows an user defined dictionary to which we can add new words to the segmentation model. However, jieba can not parse the syntax of a sentence. Other classical NLP libraries like NLTK and Stanford CoreNLP get more complicated with an user defined dictionary and do not use the results of the jieba segmented sentences in their processing. Hence HanLP was the best choice for us.

The result of a sentence dependency parsing, for the example *我验了5 号，他是张查杀" (in English: "I verified P5, he is a villain") is as below:

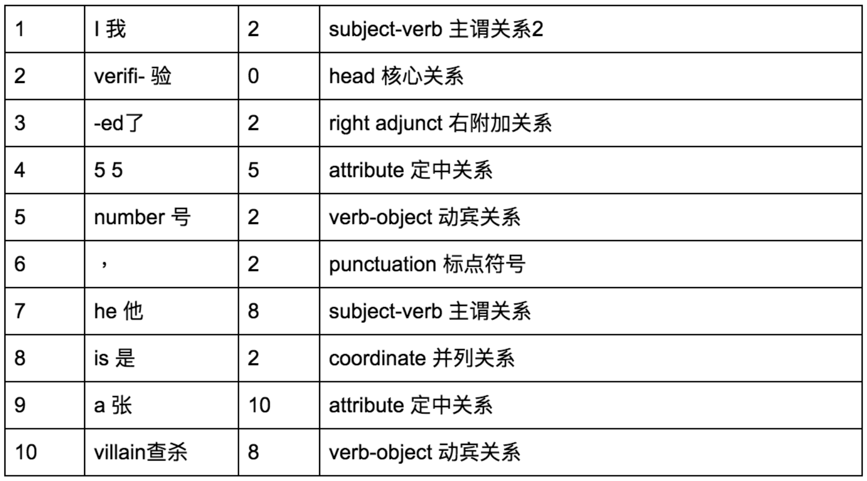
Ind	**Text**	**Lemma**	**POS**	**POS**	**Dependent**	**Dependency**
1	我	我	`r`	`rr`	2	主谓关系
2	验	验	`v`	`v`	0	核心关系
3	了	了	`u`	`ule`	2	右附加关系
4	5	5	`m`	`m`	5	定中关系
5	号	号	`q`	`q`	2	动宾关系
6	，	，	`wp`	`w`	2	标点符号
7	他	他	`r`	`r`	8	主谓关系
8	是	是	`v`	`vshi`	2	并列关系
9	张	张	`q`	`q`	10	定中关系
10	查杀	查杀	`n`	`n`	8	动宾关系

In figure 8 we translate the Chinese in literal English so that it is more clear for the readers.

It is a ConLL format object. The first column is the index of word, the second one the text, the third one its lemma, the fourth and fifth one its nature (POS tag), the seventh is the index of the word from which this word depends, and the eighth column gives their dependency relation. One can consult the complete documentation of the labels' meaning. Here we will focus on the dependency relation.

According to this table, the sentence could be seen as composed of two phrases. The head of the sentence is "verify", but since there is another verb "is" in the second phrase, it is labeled as "coordinate" to the head verb. Respectively, their subjects are 1 and the 7. Their objects are 5 and 10. The word "P5"

[18]`https://github.com/fxsjy/jieba`



1	I 我	2	subject-verb 主谓关系2
2	verifi- 验	0	head 核心关系
3	-ed了	2	right adjunct 右附加关系
4	5 5	5	attribute 定中关系
5	number 号	2	verb-object 动宾关系
6	，	2	punctuation 标点符号
7	he 他	8	subject-verb 主谓关系
8	is 是	2	coordinate 并列关系
9	a 张	10	attribute 定中关系
10	villain查杀	8	verb-object 动宾关系

Figure 8: English translation of the ConLL output example by HanLP

is segmented as 2 words in Chinese, where "号" is the object and "5" is the attribute.

For each sentence, we will construct a list keeping the most basic syntaxic information , giving two places for verbs, two places for subjects and two places for objects :

```
[’核心关系’,’核心关系’,’主谓关系’,’主谓关系’,’动宾关系’,’动宾关系’]
[’verb’,’verb’,’subject’,’subject’,’object’,’object’]
```

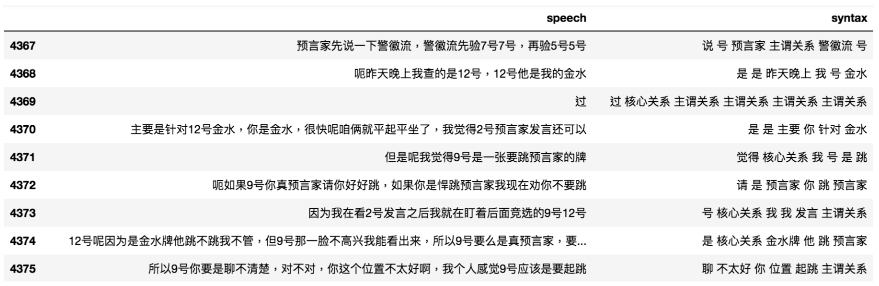
If we have a word labeled as `head` verb in the sentence, we then replace the first `’verb’` in the list with the word. If we find its `coordinate` verb, we then fill it to the second place for verbs. If there are still more verbs, we ignore them.

For the subjects, besides the words labeled as `subject-verb` dependent to the head, we also look for the word with `coordinate` relation to the subject that we have found. If there are more than 2 subjects, we ignore the latters. The objects are treated in the same way.

In our example, we get a list like this:

```
[’说’,’号’,’预言家’,’主谓关系’,’警徽流’,’号’]
[’say’,’number’,’Seer’,’subject’,’jinhuiliu’,’number’]
```

We then store them in the dataframe as a string, under the "syntax" column, as shown in figure 9.



	speech	syntax
4367	预言家先说一下警徽流，警徽流先验7号7号，再验5号5号	说 号 预言家 主谓关系 警徽流 号
4368	呃昨天晚上我查的是12号，12号他是我的金水	是 是 昨天晚上 我 号 金水
4369	过	过 核心关系 主谓关系 主谓关系 主谓关系 主谓关系
4370	主要是针对12号金水，你是金水，很快呢咱俩就平起平坐了，我觉得2号预言家发言还可以	是 是 主要 你 针对 金水
4371	但是呢我觉得9号是一张要跳预言家的牌	觉得 核心关系 我 号 是 跳
4372	呃如果9号你真预言家请你好好跳，如果你是悍跳预言家我现在劝你不要跳	请 是 预言家 你 跳 预言家
4373	因为我在看2号发言之后我就在盯着后面竞选的9号12号	号 核心关系 我 我 发言 主谓关系
4374	12号呢因为是金水牌他跳不跳我不管，但9号那一脸不高兴我能看出来，所以9号要么是真预言家，要...	是 核心关系 金水牌 他 跳 预言家
4375	所以9号你要是聊不清楚，对不对，你这个位置不太好啊，我个人感觉9号应该是要起跳	聊 不太好 你 位置 起跳 主谓关系

Figure 9: The basic syntactic structure extracted from the sentences.

Now, finally, we transform this "syntax" information into explicit information such as "who attacks whom", "who protects whom", "who defends himself against whom", etc. We proceed as follows: if we find the verbs in the dependency structure to be specific terms and the labeled intent corresponds to them, we have no doubt about the intent meaning, then we look at the object - it is generally the player number, we then take it as the target. If we find the verbs to be specific terms but the labeled intent does not correspond to them, we take the specific terms meaning as the intent meaning then retrieve the player number as the target. Else if we do not find specific verbs, we then rely on the labeled intent and retrieve the target player number. The complete set of rules can be found in the code on Github.

This information is encoded in a dictionary, whose keys are the actions (such as "attack", "claim as Seer", etc.) and the values are the objects of the actions (represented as list of player numbers). For example, for the sentence "Yesterday I checked P12, I give him my jinshui", the information to be extracted is that the speaker claims P12 is good (he "认好人" him), and more specifically this claim is through jinshui (i.e. the speaker, who previously claimed to be the Seer, says he used his special ability to find P12 is good). Therefore the dictionary corresponding to the sentence is `{'认好人：':['12'],'金水：':['12']`. In general, all the possible keys are:

自认： claim oneself to be of ... identity, the value is a list which could

have identities like "神","民","好人","预言家","女巫","禁言长老","潜行者","猎人","守卫","白痴","狼" ("divine villager", "ordinary villager", "good",
"Seer", "Witch", "Ancient", "Assassin", "Hunter", "Savior", "Idiot", "werewolf")

警徽流: the order of succession as Captain, the value is a list containing the player numbers to which the speaker will give the Captain title after checking

验: to check, the value is a list containing the player numbers that the speaker wants to check or recommends the Seer to check

不确定: not sure, the value is a list containing the player numbers that the speaker suspects but not for sure

认好人: recognize his good identity, the value is a list containing the player numbers that the speaker trusts to be good

认坏人: sure of his bad identity, the value is a list containing the player numbers that the speaker thinks for sure to be bad

不对付: not go well with..., the value is a list containing the player numbers that are in opposition with the speaker and so that the speaker has to defend himself against those players

金水: sure of his good identity proved by Seer (this identity is called jinshui, the Golden Water), the value is a list containing the player numbers that the speaker thinks for sure to be good identity because the player trusts the Seer too

预言家: recognize his Seer identity, the value is a list of the player numbers that the Speaker trusts or trusted as Seer

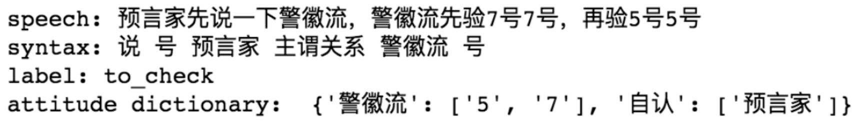
女巫: recognize her Witch identity

潜行者: recognize his Assassin identity

禁言长老:	recognize his Ancient identity
猎人:	recognize his Hunter identity
白痴:	recognize his Idiot identity
守卫:	recognize his Savior identity

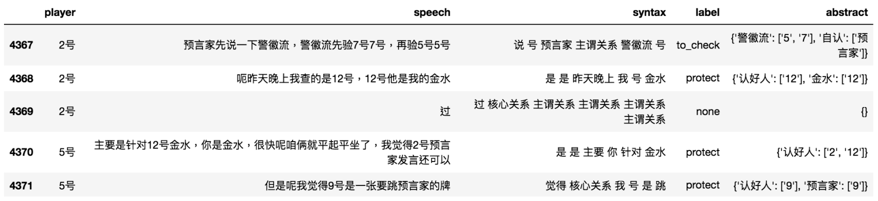
Notice that, the dictionary is for one single sentence, we will then combine them latter.

Here we just give a screenshot of the speech, syntax, label and attitude dictionary of one sentence:



```
speech: 预言家先说一下警徽流，警徽流先验7号7号，再验5号5号
syntax: 说 号 预言家 主谓关系 警徽流 号
label: to_check
attitude dictionary:  {'警徽流': ['5', '7'], '自认': ['预言家']}
```

We apply this function to every sentence, as shown below:



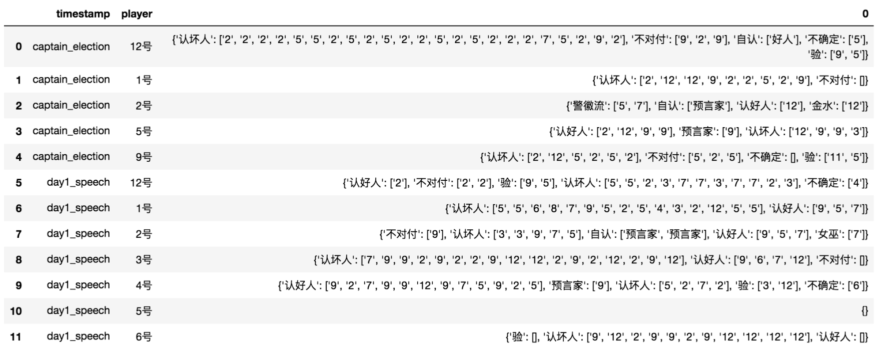
	player	speech	syntax	label	abstract
4367	2号	预言家先说一下警徽流，警徽流先验7号7号，再验5号5号	说 号 预言家 主谓关系 警徽流 号	to_check	{'警徽流': ['5', '7'], '自认': ['预言家']}
4368	2号	呃昨天晚上我查的是12号，12号他是我的金水	是 是 昨天晚上 我 号 金水	protect	{'认好人': ['12'], '金水': ['12']}
4369	2号	过	过 核心关系 主谓关系 主谓关系 主谓关系 主谓关系	none	{}
4370	5号	主要是针对12号金水，你是金水，很快呢咱俩就平起平坐了，我觉得2号预言家发言还可以	是 是 主要 你 针对 金水	protect	{'认好人': ['2', '12']}
4371	5号	但是呢我觉得9号是一张要跳预言家的牌	觉得 核心关系 我 号 是 跳	protect	{'认好人': ['9'], '预言家': ['9']}

In the following part, we summarize the attitude dictionaries (here the "abstract" column) to have a bigger view of the game situation.

2.3.b Speech turn summary

In this part we will summarize the speech of each player for each turn. As we have introduced in section 2.1.a, the csv file is ordered by speech turn in the column "timestamp", then by player number in the column "player". Hence we define a function to aggregate several attitude dictionaries from single sentences in the column "abstract" to one for each player and store it into column "0".

For the reading convenience, we display here only the columns "timestamp", "player" and "0", as in the figure 10.



	timestamp	player	0
0	captain_election	12号	{'认坏人': ['2', '2', '2', '2', '5', '5', '2', '5', '2', '5', '2', '2', '5', '2', '5', '2', '2', '2', '7', '5', '2', '9', '2'], '不对付': ['9', '2', '9'], '自认': ['好人'], '不确定': ['5'], '验': ['9', '5']}
1	captain_election	1号	{'认坏人': ['2', '12', '12', '9', '2', '2', '5', '2', '9'], '不对付': []}
2	captain_election	2号	{'警徽流': ['5', '7'], '自认': ['预言家'], '认好人': ['12'], '金水': ['12']}
3	captain_election	5号	{'认好人': ['2', '12', '9', '9'], '预言家': ['9'], '认坏人': ['12', '9', '9', '3']}
4	captain_election	9号	{'认坏人': ['2', '12', '5', '2', '5', '2'], '不对付': ['5', '2', '5'], '不确定': [], '验': ['11', '5']}
5	day1_speech	12号	{'认好人': ['2'], '不对付': ['2', '2'], '验': ['9', '5'], '认坏人': ['5', '5', '2', '3', '7', '7', '3', '7', '7', '2', '3'], '不确定': ['4']}
6	day1_speech	1号	{'认坏人': ['5', '5', '6', '8', '7', '9', '5', '2', '5', '4', '3', '2', '12', '5', '5'], '认好人': ['9', '5', '7']}
7	day1_speech	2号	{'不对付': ['9'], '认坏人': ['3', '3', '9', '7', '5'], '自认': ['预言家', '预言家'], '认好人': ['9', '5', '7'], '女巫': ['7']}
8	day1_speech	3号	{'认坏人': ['7', '9', '9', '2', '9', '2', '2', '9', '12', '12', '2', '9', '2', '12', '2', '9', '12'], '认好人': ['9', '6', '7', '12'], '不对付': []}
9	day1_speech	4号	{'认好人': ['9', '2', '7', '9', '9', '12', '9', '7', '5', '9', '2', '5'], '预言家': ['9'], '认坏人': ['5', '2', '7', '2'], '验': ['3', '12'], '不确定': ['6']}
10	day1_speech	5号	{}
11	day1_speech	6号	{'验': [], '认坏人': ['9', '12', '2', '9', '9', '2', '9', '12', '12', '12', '12'], '认好人': []}

Figure 10: Attitude dictionaries are put together in column 0.

We found that in index 10, player 5 does not have summarized dictionary, that is because he is forbidden from speaking in this turn. This information is stored apart in the behavior record about which we will explain more details in part 3.

The format of dictionary is still too noisy for the analysis. Besides, there is a lot of information redundancy. That is because during a speech turn, one may keep attacking a player with several sentences.

We then come up with the design of a "personal perspective pattern", that is to say, we will make a list of 13 items, for which the original value is "unknown". The list represents the identities of all the 12 players in the game, in the point of view of a player. The identities are the key in the summarized attitude dictionary. In Python code, the index of a list begins from zero; for us the second item, whose index is 1, will represent P1; the third item, whose index is 2, will represent P2, etc.

For example, if the personal perspective pattern of player 2 is

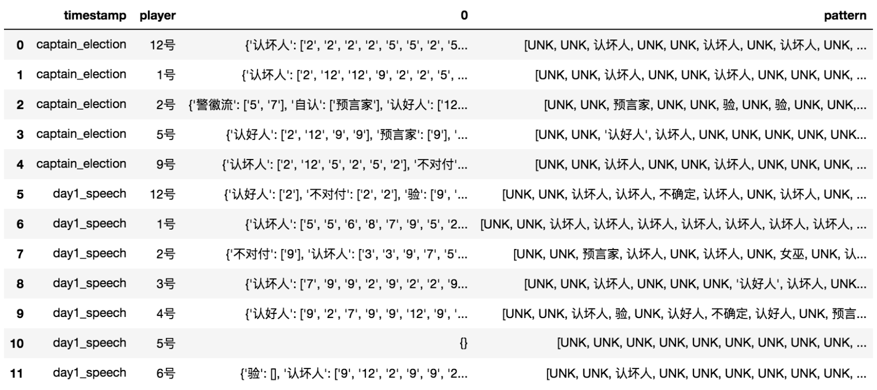
```
["UNK", "UNK", "预言家", "UNK", "UNK", "验", "UNK", "验",
"UNK", "UNK", "UNK", "UNK", "金水"]
["UNK", "UNK", "Seer", "UNK", "UNK", "Check", "UNK", "Check",
"UNK", "UNK", "UNK", "UNK", "jinshui"]
```

this means that P2 sees himself as the Seer, wants to check P5 and P7, and recognized P12 as jinshui.

We should pay attention that the keys of the attitude dictionary are not always at same level. For example, one may recognize another as "good" but also more specifically as "jinshui" (the Golden Water, the one who is checked by the Seer and verified as good). Here "jinshui" is more precise than "good", hence we will take "jinshui" as useful information instead of "good".

Another point to pay attention is that, a player may recognize another one as "good" then change his mind to think of the player as "bad" (i.e. wolf) or conversely. But since we summarize his speech in dictionary format, we do not have the information of which identity is his final decision. Fortunately we have the list of "good" and "bad", and in each list, a player may occur several times because the speaker will talk about this player several times. We then count the occurence of the player for "good" and "bad" respectively, then choose the most frequent one as his identification. It is not a completely correct solution but it is acceptable.

The final information retrieved is like below:



	timestamp	player	0	pattern
0	captain_election	12号	{'认坏人': ['2', '2', '2', '2', '5', '5', '2', '5...	[UNK, UNK, 认坏人, UNK, UNK, 认坏人, UNK, 认坏人, UNK, ...
1	captain_election	1号	{'认坏人': ['2', '12', '12', '9', '2', '2', '5', ...	[UNK, UNK, 认坏人, UNK, UNK, 认坏人, UNK, UNK, UNK, ...
2	captain_election	2号	{'警徽流': ['5', '7'], '自认': ['预言家'], '认好人': ['12...	[UNK, UNK, 预言家, UNK, UNK, 验, UNK, 验, UNK, UNK,...
3	captain_election	5号	{'认好人': ['2', '12', '9', '9'], '预言家': ['9'], '...	[UNK, UNK, '认好人', 认坏人, UNK, UNK, UNK, UNK, UNK...
4	captain_election	9号	{'认坏人': ['2', '12', '5', '2', '5', '2'], '不对付'...	[UNK, UNK, 认坏人, UNK, UNK, 认坏人, UNK, UNK, UNK, ...
5	day1_speech	12号	{'认好人': ['2'], '不对付': ['2', '2'], '验': ['9', '...	[UNK, UNK, 认坏人, 认坏人, 不确定, 认坏人, UNK, 认坏人, UNK, ...
6	day1_speech	1号	{'认坏人': ['5', '5', '6', '8', '7', '9', '5', '2...	[UNK, UNK, 认坏人, 认坏人, 认坏人, 认坏人, 认坏人, 认坏人, 认坏人, ...
7	day1_speech	2号	{'不对付': ['9'], '认坏人': ['3', '3', '9', '7', '5'...	[UNK, UNK, 预言家, 认坏人, UNK, 认坏人, UNK, 女巫, UNK, 认...
8	day1_speech	3号	{'认坏人': ['7', '9', '9', '2', '9', '2', '2', '9...	[UNK, UNK, 认坏人, UNK, UNK, UNK, '认好人', 认坏人, UNK...
9	day1_speech	4号	{'认好人': ['9', '2', '7', '9', '9', '12', '9', '...	[UNK, UNK, 认坏人, 验, UNK, 认好人, 不确定, 认好人, UNK, 预言...
10	day1_speech	5号	{}	[UNK, UNK, UNK, UNK, UNK, UNK, UNK, UNK, UNK, ...
11	day1_speech	6号	{'验': [], '认坏人': ['9', '12', '2', '9', '9', '2...	[UNK, UNK, 认坏人, UNK, UNK, UNK, UNK, UNK, UNK, ...

In the next part, we will try to guess which one is most likely to be the werewolf, with help of the behavior information.

3 Behavior processing

What is behavior in the game ? As we explained in the game rules section 1.3, in Day 1 there is at first a captain election, the elected captain will have a 1.5 weight in the subsequent votes for execution. After the captain election, the Moderator will announce the death information in Night 1. Then the Captain will choose a player to start the speech tour. After everyone has spoken, all the players will vote for someone's execution. All these information, including election results, votes, captain's decision, along with the werewolf's suicides (if any), are considered as behavior.

Novice players often have nothing to talk about, while in this professional-grade game, players often have too much to say. In fact, when we study the content and meaning of the speech, we found these professional players tend to analyze the possible character of one player by three aspects: his statement, his behavior and his general attitude. For the statement, besides the meaning of the sentence, the expression that he uses and even the emotion that he shows also count as very important. As for the behavior, let's provide an example. Suppose in the captain election, a werewolf A pretends to be the Seer and he declares having identified werewolf B as bad, and there are two other candidates including a villager and the real Seer who declares that A is a werewolf. If the werewolf B votes for the villager to be Captain, this behavior makes B to expose himself, for the reason that a villager will certainly vote for the Seer who declares bad the Seer who declares him bad. This shows how the behavior is sometimes revealing. The general attitude that the player exudes is the last thing that we are interested in, it is useful only when the game is face to face or the players know each other. We often talk about the normal performance of someone and the abnormal aspects that he has in a particular game.

3.1 Behavior record

Here we give the list of behavior features that might be taken for considerations in the future prediction of player's character, ordered by the turn. We take the game S1E101 as example.

Initial parameters:

- `number_of_players: 12`
- `part-to-kill: True`
- `character_pattern: {'n_villager': 4, 'n_god': 4, 'ordinary_werewolf': 4, 'white_wolf': 0, 'seer': 1, 'witch': 1, 'hunter': 0, 'idiot': 0, 'ancient': 1, 'assassin': 1, 'savior': 0}`
- `witch_can_save_herself: True`
- `character_distribution: ['UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK', 'UNK']`
- `player_alive: [‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’]`

The `character_pattern` is the game pattern. It is represented by a dictionary where the key is a character (or role) and the value is the number of players having that role.

The `character_distribution` is a list which contains 13 items. The item represents the known characters, where the index of an item is the number the corresponding player player. We don't have a "player 0" so the first item of the list, which has index 0, is just unknown and serves no purpose.

Notice here that, the `character_distribution` list is different from the `personal_perspective_pattern` from the previous part. The `character_distribution` represents the reality, it can only note the certainly known players' character such as a suicided werewolf, a Hunter who fires or an idiot who is executed but can stay in the game.

The `player_alive` also has 13 items. It represents the life status of each player. In the initial status, everyone is alive.

Captain election:

- `candidates: [‘1’, ‘2’, ‘5’, ‘9’, ‘12’]`
- `non_candidates: [‘3’, ‘4’, ‘6’, ‘7’, ‘8’, ‘10’, ‘11’]`
- `speech_order: [‘2’, ‘5’, ‘9’, ‘12’, ‘1’]`
- `still_candidates: [‘2’, ‘9’]`
- `order_succession_captain: {‘2’:[‘7’, ‘5’], ‘9’:[‘5’, ‘11’]}`
- `votes: {‘3’: none, ‘4’: none, ‘6’: ‘9’, ‘7’: ‘9’, ‘8’: ‘9’, ‘10’: ‘9’, ‘11’: ‘9’}`
- `character_distribution: [’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’]`
- `player_alive: [‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘Captain’, ‘alive’, ‘alive’, ‘alive’]`

The lists `candidates` and `non_candidates` record who participated in the election and who didn’t. In the first vote, only the `non_candidates` can vote. If there’s a tie, all the players except the tied candidates can vote. This can repeat once more before abandoning the election process (without Captain).

The `speech_order` is also important. Once someones has spoken, he can not speak again in this turn. So if a player claims himself to be the Seer, and he declares a accuses in a former position to be bad, then the former player can no longer argue for himself.

The `still_candidates` is the list of the players who are still candidates after every candidate has spoken (i.e. those who did not quit the election). The Captain should be one of these players.

The order of succession as Captain is an attribute that we have introduced in the previous part. We note this down for further analysis.

The dictionary votes records everyone's vote. "None" indicates that P3 and P4 abstained.

In `player_alive`, we put the item of index 9 to be "Captain" to record that he is the Captain.

Day1 speech:

- `night_death: [ ]`
- `forbidden: [‘5’]`
- `speech_order: [‘8’, ‘7’, ‘6’, ‘5’, ‘4’, ‘3’, ‘2’, ‘1’, ‘12’, ‘2’]`
- `call_vote: [ ]`
- `votes: {}`
- `character_distribution: [’UNK’, ’UNK’, ’werewolf’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’]`
- `player_alive: [‘alive’, ‘alive’, ‘suicide’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘alive’, ‘Captain’, ‘alive’, ‘alive’, ‘alive’]`

The `forbidden_speaking` is in this pattern specifically because in this game we have "ancient" who can forbid one player from speaking during one turn.

Note that in this example there is no vote, since one werewolf suicided (as we can see in `character_distribution`) hence stopping the process.

Day2 speech:

- `night_death: [‘3’, ‘9’]`
- `forbidden: [‘4’]`
- `speech_order: [‘6’, ‘7’, ‘8’, ‘10’, ‘11’, ‘12’, ‘1’, ‘5’]`
- `call_vote: [ ]`

- `votes: {‘1’: ‘6’, ‘5’: ‘8’, ‘6’: ‘6’, ‘7’: ‘8’, ‘8’: ‘6’, ‘10’: ‘11’, ‘11’: ‘6’, ‘12’: ‘6’}`
- `character_distribution: [’UNK’, ’UNK’, ’werewolf’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’, ’UNK’]`
- `player_alive: [‘alive’, ‘alive’, ‘suicide’, ‘eliminated’, ‘alive’,‘Captain’, ‘executed’, ‘alive’, ‘alive’, ‘eliminated’, ‘alive’, ‘alive’, ‘alive’]`

The `call_vote` is to note down the call of the Captain, he could call everyone to vote for some players’ execution. His vote count 1.5 points.

To adapt the syntax of json format, we should keep every dictionary with the same keys, so the final structure of behavior information is like below:

```
{
  "stage":"day2_speech",
  "dispositive":{
    "character_pattern": {"n_villager": 4, "n_god": 4,
    "ordinary_werewolf": 4, "white_wolf": 0, "seer": 1, "witch":
    1, "hunter": 0, "idiot": 0, "ancient": 1, "assassin": 1,
    "savior": 0, "part-to-kill":"True",
    "witch_can_save_herself":"True"},
    "night_death":["3","9"],
    "forbidden":["4"],
    "candidates": [],
    "non_candidates":[],
    "speech_order":["6", "7", "8", "10", "11", "12", "1", "5"],
    "still_candidates": [],
    "order_succession_captain": [],
    "call_vote":[],
    "votes": [{"1": "6", "5": "8", "6": "6" ,"7": "8", "8": "6",
    "10": "11", "11": "6", "12": "6"}],
    "player_alive": ["alive", "alive", "suicide", "eliminated",
    "alive", "Captain", "executed", "alive", "alive",
    "eliminated", "alive", "alive", "alive"],
    "character_distribution":["UNK", "UNK", "werewolf", "UNK",
    "UNK", "UNK", "UNK", "UNK", "UNK", "UNK", "UNK", "UNK",
    "UNK"]
  }
},
```

The complete behavior records can be found in the Github page.

In the next part, we will try to guess which players are the werewolves, with help of the retrieved speech information plus the behavior.

3.2 Who's the werewolf

In this part, we will use all the information extracted in the previous parts to make an educated guess about who the werewolves are.

A first idea is to assign each player a base likelihood of $\frac{1}{3}$ (since we want 4 werewolves among 12 players) and increment or decrement these likelihood probabilities according to the information we get.

The rules could be as below:

- Someone who is suicided is certainly a werewolf, so we should raise his probability to 1.
- In a speech turn, if one player claims himself as the Seer and gives another player a checked good identity, but the latter is proved to be a werewolf, then the probability of the former should also be 1.
- In a speech turn, those who are marked bad by all the players who claim themselves as the Seer (in particular by the real Seer) are certainly werewolves, their probability should be 1.

- In a speech turn, especially in the captain election, if two players claim themselves to be Seers, then there must be a true Seer and a werewolf.
- In a speech turn, the one who receives most attacks should get an increment in a speech turn, the one who receives most protects should get a decrement.
- In a speech turn when vote for execution, the one who receives most votes should get a greater increment.
- In a speech turn when vote for captain election, the one who receives most votes should get a decrement.

These rules make some sense but they have some issues. Rules 5-8 directly follow the general sentiment of the players, but the players are often misguided; rules 1-3 rely on events that are too rare. Instead, of directly following other players, we want to measure what their sentiment and their subsequent actions tell about them. For example, if a player in is widely suspected in a game to be a wolf, this does not necessarily mean he is; however it makes it suspicious for other players to support this player. Also, the wolves have a tendency to protect themselves. To incorporate this kind of reasoning, we need to consider the wolves not individually but collectively.

Therefore the basic idea of our algorithm will be the following. Our goal is to find 4 werewolves among 12 players, so we look for the likelihood of the Team Werewolf to be some fixed combination. That is to say, to all the $\binom{12}{4} = 495$ combinations of 4 players, we will assign a "penalty score" that should be higher when the probability of Team Werefolf being said combination is lower. We do this by listing a set of rules: each rule should filter a scenario where the speech and behavior information somewhat contradicts the hypothesis that Team Werewolf is considered combination of four players, then increment the penalty score of that combination.

More specifically, we consider a list `wolf_list` of four players and its complementary `villager_list` in `range(1,13)`. We will make the hypothesis that these lists accurately represent the situation and test them against the data.

The guiding principle is that the wolves know each other hence should not accuse themselves of being bad, nor claim villagers to be good. However, following such a strategy would rapidly make the other players notice the pattern and raise suspicion. Consequently, wolves will often find a "scapegoat" among them and abandon him. This way, if the scapegoat is revealed to be indeed a wolf, other wolves who had accused him would look good in the eyes of everyone.

We translate this thinking by first writing a function `find_scapegoat` that assigns to each item in `wolf_list` a score proportional to the number of execution votes or accusation speeches received by other players in `wolf_list`. The output of `find_scapegoat` is then the item with maximum score.

Now, integrating speech and behavior information only from the suspected wolves would not be enough. The problem is that the villagers are mostly in the dark because the roles are kept secret. A way of remedying this is to have lists of players `probably_good` and `probably_bad` that try to get the consensus emerging from the speeches. Then, a villager would mostly want to support the `probably_good` players and denounce the `probably_bad` players. The attitude of the wolves towards these players is less informational: they have incentives both to support or denounce those players.

We build the list `probably_good` as the list of players who have received approval from a number of players greater than a certain threshold, and similarly `probably_bad` is the list of players who have been denounced by a number of players greater than another threshold.

Now, we have what we need to compute the score of the list `wolf_list` (more accurately, the score of the set set(`wolf_list`)). We start with `score = 0`, and increment according to the following rules:

- If there is one player that is marked as wolf in `character_distribution` (i.e. he suicided, so we are sure of this information) but is not a wolf under our hypothesis, we raise `score` to a very large value `infty`.

- For each (hypothesised) wolf that denounces another wolf that is not the scapegoat, we increase `score` by a constant `alpha`. If this denunciation is backed up by a vote for execution, we further increase by another constant `alphavote`.

- For each wolf that supports a villager, we increase `score` by `beta`. If this support is backed up by a vote during the Captain election, we further increase by `betavote`.
- For each villager that denounces a player in `probably_good`, we increase the score by `gamma`. Similarly, we also have `gammavote`.
- For each villager that supports a player in `probably_bad`, we increase the score by `delta`. Similarly, we also have `deltavote`.

Further, the increments in score that result from the speech or behavior of a wolf that is confirmed by `character_distribution` (he is absolutely a wolf) are multiplied by a coefficient `absolute_coef`.

Also, we need to keep track of the time. Indeed, the more we advance in the game, the more the players know so the more their decisions carry weight. Therefore, we have a parameter `time_decay`, and the score increments that result from an action (speech or behavior) taken during the day `t` speech round is multiplied by `time_decay**t` (the Captain election speech round corresponds to `t=0`). The `probably_good` and `probably_bad` lists are also updated for each round, to account for the fact that they are built from the ever-increasing information the villagers get.

When all this is done, we aggregate the scores of each of the 495 possible sets `set(wolf_list)`. The wolf team should be the one having the lowest score.

3.3 Results analysis

For the evaluation of this predictor, we test on the three games S1E101, S1E102 and S1E103. In order for the evaluation to be impartial, for each evaluation game we retrained the SVM classifier on the sentences coming from the eight other games of the corpus only.

S1E101 We show the result of S1E101's Werewolf Team's prediction and the most likely top 10 combinations with their penalty score as below. This was computed using the data of the whole game.

wolf team	**score**
(2, 3, 4, 9)	15.54
(1, 2, 3, 10)	16.0
(2, 3, 9, 10)	16.14
(2, 5, 9, 11)	17.14
(1, 2, 5, 11)	18.9
(2, 3, 5, 11)	18.9
(2, 3, 4, 10)	18.98
(2, 3, 4, 11)	20.0
(2, 4, 9, 11)	20.02
(2, 5, 7, 11)	20.29

In this game, P2, P3, P4 and P11 were the real 4 werewolves. We are glad to see P11 occur many times even if he is not put in the most probable werewolf team. But even though our predictor guesses 3 werewolves in 4, the player P9 is put also in the combination and also appears repeatedly in the other probable combinations. P9 was in fact the Seer so he was from the beginning in the opposite team.

We can see how the predicted most probable werewolf team evolves as the predictor gets progressively, turn by turn, the information from the game each:

turn	**predicted wolf team**	**score**
`captain_election`	(1, 3, 4, 9)	0.0
`day1_speech`	(2, 4, 8, 12)	5.62
`day2_speech`	(1, 2, 4, 8)	9.12
`day3_speech`	(2, 3, 4, 9)	15.54

To account for the imprecision of the method, we also compute a "probability" for each individual player to be a werewolf by counting the occurrences in the top $N = 10$ combinations. The lists of the probabilities (player `i` in index `i-1`) are given here for each speech turn (and we remind in the last column of the top scoring combination) :

turn	**individual wolf probabilities**	**predicted individual wolves**	**predicted wolf team**
`captain_election`	[0.2, 0.1, 1.0, 0.8, 0, 0.3, 0.2, 0.2, 0.8, 0.1, 0.1, 0.2]	(3,4,9,6)	(1, 3, 4, 9)
`day1_speech`	[0, 1.0, 0, 0.8, 0, 0.1, 0.2, 0.7, 0, 0.4, 0.3, 0.5]	(2,4,8,12)	(2, 4, 8, 12)
`day2_speech`	[0.3, 1.0, 0.3, 0.8, 0, 0.1, 0.1, 0.6, 0.3, 0.3, 0, 0.2]	(2,4,8,1/3/9/10)	(1, 2, 4, 8)
`day3_speech`	[0.2, 1.0, 0.6, 0.4, 0.4, 0, 0.1, 0, 0.4, 0.3, 0.6, 0]	(2,3,11,4/5/9)	(2, 3, 4, 9)

In general, individually, at the end the player P2, P3, P11 are the most probable werewolves, the combination of P2, P3, P4 and P9 is the most probable werewolf team.

The reality is that P2, P3, P4 and P11 were the real 4 werewolves, among which P2 suicided at the end of `day1_speech`, P3 was poisoned by the Witch and P4 was alive until the final turn. It is satisfying to see that our predictor is capable to find the werewolves who did not suicide like P3. We are also glad to see that our predictor has found P4 very early from the beginning to the end, faring even better than human players.

S1E102 The top predicted combinations at each turn are:

turn	**predicted wolf team**	**score**
`captain_election`	(1, 2, 3, 5)	0.0
`pk_speech`	(2, 3, 10, 11)	0.0
`extra_pk`	(2, 4, 10, 11)	20.84
`day1_speech`	(1, 4, 10, 11)	24.25
`day1_lastwords`	(1, 2, 4, 11)	27.79
`day2_speech`	(1, 4, 10, 11)	24.25
`day3_speech`	(8, 9, 10, 11)	34.53
`day4_speech`	(1, 9, 10, 11)	36.67

The top predicted individuals for each turn, aggregated from the top 10 combinations, are:

turn	individual wolf probabilities	predicted individual wolves	predicted wolf team
captain_election	[1.0, 1.0, 0.4, 0, 0.4, 0, 0, 0, 0.4, 0.4, 0.4, 0]	(1,2,3/5/9/10/11)	(1, 2, 3, 5)
pk_speech	[0, 0.6, 0.4, 0.3, 0.5, 0.5, 0.1, 0.5, 0.2, 0.5, 0.4, 0]	(2,5/6/8/10)	(2, 3, 10, 11)
extra_pk	[0.4, 0.4, 0.4, 1.0, 0.4, 0, 0, 0, 0, 0.4, 1.0, 0]	(4,11,1/2/3/5/10)	(2, 4, 10, 11)
day1_speech	[0.6, 0.5, 0.3, 0.8, 0.1, 0.1, 0, 0.1, 0.2, 0.4, 0.9, 0]	(11,4,1,2)	(1, 4, 10, 11)
day1_lastwords	[0.7, 0.6, 0.6, 0.7, 0.7, 0, 0, 0, 0, 0, 0.7, 0]	(1,4,5,11)	(1, 2, 4, 11)
day2_speech	[0.2, 0.2, 0.1, 0.9, 0.2, 0, 0.1, 0.1, 0.1, 1.0, 0.9, 0.2]	(10,4,11,1/2/5/12)	(1, 4, 10, 11)
day3_speech	[0.1, 0.1, 0.1, 0.2, 0.1, 0, 0.1, 0.2, 1.0, 1.0, 0.8, 0.3]	(9,10,11,12)	(8, 9, 10, 11)
day4_speech	[1.0, 0.2, 0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.9, 0.9, 0.1, 0.1]	(1,9,10,2/3/4)	(1, 9, 10, 11)

The real 4 werewolves in S1E102 were P1, P3, P9 and P10. Among them, P10 suicided in `day2_speech`, P9 suicided in `day3_speech`, P1 suicided at the end. Our predictor has successfully found P1, P3 and P10 early before they suicided, even better than a part of human players.

S1E103 The top predicted combinations at each turn are:

turn	predicted wolf team	score
captain_election	(1, 4, 6, 7)	0.0
day1_speech	(2, 6, 8, 12)	16.14
day2_speech	(2, 3, 8, 12)	23.628

The top predicted individuals for each turn, aggregated from the top 10 combinations, are:

turn	individual wolf probabilities	predicted individual wolves	predicted wolf team
captain_election	[0.3, 0.7, 0.1, 0.8, 0.3, 0.5, 0.3, 0, 0.6, 0.3, 0, 0.1]	(4,2,9,6)	(1, 4, 6, 7)
day1_speech	[0.1, 0.4, 0.3, 0.4, 0.3, 0.5, 0, 1.0, 0.2, 0, 0, 0.8]	(8,12,6,2/4)	(2, 6, 8, 12)
day2_speech	[0.1, 0.3, 1.0, 0.1, 0.1, 0.4, 0, 1.0, 0, 0.1, 0.3, 0.6]	(3,8,12,6)	(2, 3, 8, 12)

The real werewolves were P3, P8, P11, P12. Among them, P8 suicided in the `day1_speech` and P3 suicided in the `day2_speech`, P11 and P12 were eliminated during the night. Our predictor has successfully found the hidden wolf P12 but failed to find P11.

4 Discussion

In our project, we have built a simple model to predict the werewolves' identities in the game of the Werewolves of Millers Hollow in order to help build a chatbot which could react as a real player in the future. We began by simply suppose that we are in an ideal situation, that is to say, all the input is in form of text and we play as a villager which only has a task of analyzing, responding and voting. In this paper, we merely focus on the analyzing part of the chatbot.

In the first step, we tried several machine learning and deep learning methods by Scikit-Learn and Keras to classify the intent of a single sentence on a manually annotated dataset, then finally chose the SVM model. In the second step, we got help from the Chinese dependency parsing tool HanLP to extract sentence main parts and to retrieve concrete information for later step. Finally, using all this extracted information, we simulated the probable werewolves combinations then assigned scores to all these hypothetical combinations by hand-crafted rules, to find the combination of 4 werewolves which is the most likely to be validated.

We are far from building a real chatbot AI player in this game. Besides working to build the response and vote parts, we already have much room for improving our game analysis model. For example, the RNN and LSTM model costs the most time but receive worst results, which we totally did not expect. We could add more syntactical features to LSTM model or use word embedding representation in the other methods. Moreover, the precise comprehension of single sentence has also many errors, especially when facing ambiguous sentences. Another area of improvement is the hand-crafted rules in the last part. They stem from a very simple modelisation of the game's most basic and naive strategies: we did not even go beyond wolves and villagers, when there are strategies that revolve around the Seer for example (especially in the Captain election round) that reveal a lot about the players' team.

Despite all these over-simplifications, we found on our three test games that the prediction is surprisingly good, in fact better than most of the human players in those game (and those were seasoned Werewolf players!). We expect that exploring the numerous possibilities of improvement could yield even more remarkable results.

All comments are welcome on the Github page of the project, where all the data and scripts are also available: `https://github.com/ExeCuteRunrunrun/loup-garou`

Appendices

A Tools and libraries

The following tools and libraries were used for this project:

Python 3.6.3

ipython 6.1.0

jupyter 1.0.0

HanLP 0.1.41 (python interface pyHanLP)

jieba 0.39

Keras 2.0.6 (with tensorflow 1.0.0 back end)

Pandas 0.20.3

Scikit-Learn 0.19.1

B Glossary of Werewolf game-specific expressions

Here is a list of the most common expressions in use during the in-game discussions, along with their pinyin pronunciation, their literal translation and their definition.

神民：**shenmin** divine villagers; villagers who have special abilities

狼人/狼/匪：**langren/lang/fei** werewolves; werewloves

自爆：**zibao** explode; werewolf suicide

公投权：**gongtouquan** public vote; vote to decide who is to be executed

警徽流：**jinghuiliu** order of succession as Captain; if the Captain is the Seer, he can verify the identity of one person every night, if this person is good, he will pass his badge to this person when he dies, and if not, to the next one in the jinghuiliu

上警：**shangjing** go to captain election; participate in the captain election

退水：**tuishui** go back to water; quit the election

警上：**jingshang** in the election; ex. there are now two Seers in the election

警下：**jingxia** out of the election; the players who didn't participate in the election from beginning

归票权：**guipiaoquan** right of reuniting votes; call the others to vote against someone

验/验人/查验：**yan/yanren/chayan** verify identity; verify a player's identity during the night to know wether he's good or bad

查杀：**chasha** verify kill; verify someone's idententity to discover he's a wolf

金：**jinshui** golden water; good identification, verify someone to discover he's a villager

银水：**yinshui** silver water; the one who is saved by the Witch (hence he is probably good, except if he is a suicide wolf)

铜水：**tongshui** copper water; the one who is protected by the Savior

挂身份：**guashenfen** hang on his identity; might have a functional identity, ie divine villager or werewolf

跳/起跳：**tiao/qitiao** jump out; claim oneself to be some divine villager (eg. tiao the Seer means to claim oneself as the Seer)

穿衣服：**chuanyifu** wear the cloth of...; declare oneself as some divine villager

悍跳：**hantiao** bold jump; when a wolf declares himself as a divine villager, often the Seer

冲/冲票/绑票：**chong/chongpiao/bangpiao** bind the votes; since the werewolves are a group, they can reunite all their votes to vote someone to be executed, it is risky behavior because they may then be exposed

民及民以下：**minjiminyixia** villager and below villager; equal to or worse than ordinary villager, ie either an ordinary villager or a werewolf

狼面：**langmian** werewolfness; the probability of being a werewolf

聊爆：**liaobao** explode by talking; a werewolf who exposes himself by mistake

深水狼：**shenshuilang** deep water wolf; a werewolf who hides very well

铁狼：**tielang** iron wolf; someone who is certainly a wolf

金刚狼：**jinganglang** diamond wolf; a werewolf whom everyone thinks confidently is good

隐狼：**yinlang** hiding wolf; a hiding werewolf

怂狼：**songlang** weak wolf; a werewolf who is unconfident and afraid, a game where no wolf participates in the captain election is also called a weak game

刀：**dao** knife; kill

挨刀：**aidao** get killed

自刀：**zidao** kill oneself; when the werewolves at night choose to kill one of them so that he could get the Witch to save him and make everyone think he is good

卖队友：**maiduiyou** sell out one's companions; a werewolf wo denounces the other wolves in order to appear as good

闭眼玩家：**biyanwanjia** player with closed eyes; the players who do not open their eyes in the night, ie the ordinary villagers

反水：**fanshui** back water; when player A claims B to be good (through jinshui), but player B then publicly suspects A to be a werewolf

反水立警：**fanshuilijing** back water and support the Captain; to fanshui and support another candidate to the captain election

表水：**biaoshui** express water; claim oneself to be good

划水：**huashui** paddle water; have nothing important to say as if knowing nothing

排水：**paishui** drain water; guess all the good identities (and the wolves by elimination)

愚/鱼/愚民：**yu/yu/yumin** silly/fish/silly villager; the villager who does not trust the real divine villagers and who helps the werewolves

暴民：**baomin** mob; the villager who votes for the real divine's execution, too confident in his wrong judgement

警左/警右：**jingzuo/jingyou** left/right hand of the captain; the Captain chooses the order of speeches, from his left hand then clockwise, or his right hand then countclockwise

对跳：**duitiao** jump against each other; two players claim a same divine identity

扛推：**kangtui** unite everyone to push out; the werewolves accuse a villager to be bad, and all the others don't trust this player either, so the werewolves succesfully call the villager to be executed

拍/踩/打：**pai/ca/da** criticize; accuse someone or mark him as werewolf

出：**chu** out; vote someone out

挂：**gua** hang on; temporarily label someone as an option for the execution vote

飞：**fei** fly on; decide to vote for/against someone

前置位/后置位：**qianzhiwei/houzhiwei** previous/next in speech order

站边：**zhanbian** stand by one side; support one player

和认识：**renshi** know each other; ... and ... are wolves (only wolves can see each other's identity)

吃信息：**chixinxi** eat information; get information in the night, so that either he is a divine villager, or a werewolf

做好/做坏：**zuohao/zuohuai** make good/bad; take an action that makes someone seem good/bad

心路历程：**xinlulicheng** one's various thoughts (during the game)

自证身份：**zizhengshenfen** prove one's good identity; use one's special ability to prove one's good identity (only divine villagers can do this)

焦点牌：**jiaodianpai** focus point; the most controversial player

Bibliography

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[Bengio et al., 1993] Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1183–1188. IEEE.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[DeVault et al., 2014] DeVault, D., Artstein, R., Benn, G., Dey, T., Fast, E., Gainer, A., Georgila, K., Gratch, J., Hartholt, A., Lhommet, M., et al. (2014). Simsensei kiosk: A virtual human interviewer for healthcare decision support. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1061–1068. International Foundation for Autonomous Agents and Multiagent Systems.

[Fan et al., 2008] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874.

[Hansen and Salamon, 1990] Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001.

[Kannan et al., 2016] Kannan, A., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G., Lukacs, L., Ganea, M., Young, P., et al. (2016). Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 955–964. ACM.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). Scoring, term weighting and the vector space model. *Introduction to information retrieval*, 100:2–4.

[McCallum et al., 1998] McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

[McTear et al., 2016] McTear, M., Callejas, Z., and Griol, D. (2016). *The conversational interface: Talking to smart devices.* Springer.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Ng, 2018] Ng, A. (2018). Stanford CS229 Lecture Notes, Part V. URL: `http://cs229.stanford.edu/notes/cs229-notes3.pdf`. Last visited 09/2018.

[Ng and Jordan, 2002] Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848.

[Ritter et al., 2011] Ritter, A., Cherry, C., and Dolan, W. B. (2011). Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pages 583–593. Association for Computational Linguistics.

[Schneider, 2005] Schneider, K.-M. (2005). Techniques for improving the performance of naive bayes for text classification. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 682–693. Springer.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Weizenbaum, 1966] Weizenbaum, J. (1966). ELIZA - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

[Xu et al., 2017] Xu, A., Liu, Z., Guo, Y., Sinha, V., and Akkiraju, R. (2017). A new chatbot for customer service on social media. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3506–3510. ACM.