Who Is Undercover

Yuyan Zhou | Fengming Zhu | Ming Zhong | Yucong Zhang
{zhouyy1, zhufm, zhongming, zhangyc}@shanghaitech.edu.cn
School of Information Science and Technology
ShanghaiTech University
$Jan.18^{th}$

***Abstract*—In this project, we are going to design and implement an AI-agent who is able to play the game "Who is Undercover" with human players. We teach the AI-agent to understand the game process generally by building a hidden markov model. In this paper, we will follow this routine. First, we'd like to give some introduction on the background about both the AI and the game itself. Then we'd like to talk about how we model the word space, which serves as the most critical basis in our project. Then it comes to the Gaussian initialization and the kernel modeling part. In this part, we will go through the transition models, the emission models and the techniques that we have used to do the sampling.**

Keywords –Gaussian Distribution, Hidden Markov Model, Lexical semantics, Pragmatics

I. Introduction

[Overview] With the rapid development of artificial intelligence, more and more amazing AI players are created such as the most famous AlphaGo. AI players have both theoretical and practical value to explore, because it integrates a lot of advanced techniques in different subfields of AI, such as natural language processing, game theory, computer vision, etc. We try to implement this project generally using the course-related AI knowledge and techniques such as hidden markov model, sampling and natural language processing.

[Game Introduction] Who is Undercover is a game. This game needs at least three players in total. In the beginning, every player in the game is given a same word except one who gets a different but similar word. In each round, the players need to give a word about their word in hand. After each round, the players are asked to point out the one who is most likely to be the undercover, and the player who gets the most votes is out. The game ends if either the undercover is out or there are only 2 players left. Intuitively, the aim for the undercover in this game is to survive as long as possible, and the aim for the civilian is to find out who has a different word.

[AI Introduction] Our AI is designed to play *Who is Undercover* with human beings. It has a data set, which is obtained through *Word2Vec*. In order to play well in this game, our AI has to be able to accomplish three tasks. First, it has to analyze the ornamental or similar words given by other human players. Here we use the idea of *Gaussian Distribution*. Next, it has to tell who is undercover by our own AI-thinking algorithm. Finally, it has to give a particular word about the word picked in the first round. For the last two tasks, we use the idea of *Hidden Markov Model*.

II. Data set building

[Word2vec] Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. Especially, this proximity can be quantified by computing the cosine-distance of given vectors and the value is simple to compute. So we can use it to demonstrate the similarity between given words. In our game, this function is quite critical if the AI wants to "understand" the words given by the other players and then identify the identity of different players. The intuitive way is to query the similarity.

Word2vec has two main model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words. For our word space system, we use an embedded data set in a package which has already been trained well.

[Gensim] The most fundamental tool in this project is *Gensim*. Gensim is a free Python library designed to automatically extract semantic topics from documents, as efficiently and painlessly as possible. The algorithms in Gensim, such as Word2Vec, FastTex, etc, automatically discover the semantic structure of documents by examining statistical co-occurrence patterns within a corpus of training documents. These algorithms are unsupervised, which means no human input is necessary it only needs a corpus of plain text documents.

Once these statistical patterns are found, any plain text documents (sentence, phrase, word) can be succinctly expressed in the new, semantic representation and queried

for topical similarity against other documents (words, phrases).

Gensim also provides a lot of useful and convenient interfaces for users. For example, we can use an embedded data set which has already been trained well to represent our word space, which means words have already been converted into word vectors in some word space, and become . Each time we run the game, we can use "load" function to load the system from disk to memory. Then we can call the "similarity" function to compute the cosine-distance between given words. This will be helpful if the AI is trying to "understand" the word. For simplicity, we model the understanding process by computing the similarity between the new-coming information we get and the knowledge we already have. The basic idea is higher similarity indicates a closer relationship. The AI are more likely to consider the player who are giving a word with higher similarity as its companion. By contrast, lower similarity is more likely to indicate an opponent identity. We can also call the "most_similar" function to get the most n similar words and their similarity value between the input. This is useful when we build Gaussian models.

III. Gaussian initialization

This is a special case where the AI-agent is the first one to deliver some descriptor about his item in hand. At that time, the knowledge base of this AI-agent is very naive, so we have to build an initial distribution for him to let him not lose his game in the very first round.

Note that he has been given an item (or word) in the beginning of the game. We invoke the function *most_similar()* to get a bunch of similar words and their similarities represented by the cosine values between the corresponding word vectors, for each $(word, cos\theta)$ pair, we induce another pair $(word, 2 - cos\theta)$. Simply by a symmetric operation, we make the mean of those cosine values be 1. After that we build up a Gauss distribution with mean and std of all those cosine values (shown in Fig.1). And by this operation, the more similar a word is to the item-in-hand, the more possible it will be to be sampled out.

It should be clear that, in the first round, the the sampling interval is supposed to lie not that close to 1. Since the AI-agent has not enough information about other players, it not wise to deliver some descriptor very representative to the item-in-hand.

IV. HMM

As a player, AI-agent is supposed to have a perception on the current state. In other words, he should consider whether he or someone else is undercover.We implement this function by constructing a Hidden Markov Model (HMM), as is shown in Fig.2. The horizontal sequence is called a transition model, while the vertical sequence is called an emission model. Since the transition model is simple, which only contains the action of voting out a player each round, we will explain this briefly. However, in the following passage, we will focus on the

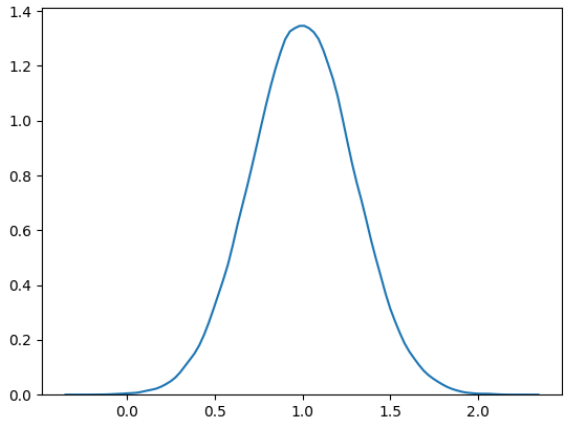

Fig. 1. Gauss initialisation

emission model, where we update our belief based on the evidence, and also the technique we have used when we are facing the sampling problems.

Hidden variable at each state is a vector including a set of probabilities. Basically, the size of this set equals to the number of players playing in the game. Accordingly, each element in the vector is relevant to the probability of he or she being undercover. We call this vector a 'belief' vector. Additionally, evidence at each state is presented by a vector, called 'evidence' vector, which contains the words that the players gave in that round. The size of this evidence vector is equal to the number of players who still survive in the game. The 'belief' vector and the 'evidence' vector are shown in Fig.3.

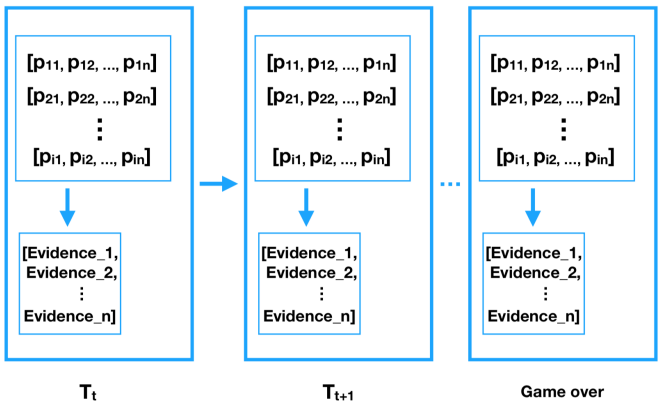

Fig. 2. The whole HMM frame

[Emission model] In our emission model, AI has to do both the observation and the updating. Generally speaking, in each round, after our AI gets the words from other players, he has to change his judgement of other players that whether they are undercover or not. Here is how we implement this.

First the observation part. There are two vital auxiliary

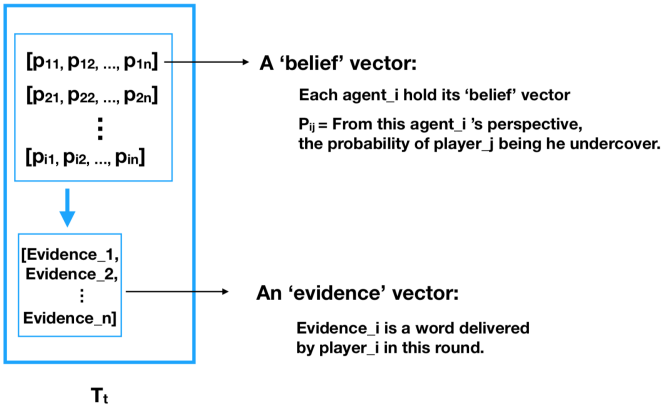

Fig. 3. One single timestamp

variables are used in our implementation, p_1 and p_2. p_1 is the probability of AI-agent being undercover. In other words, from AI-agent's perspective, it indicates how likely that he is undercover. p_2 is a set of probabilities which has the size of the number of players still alive in the game currently. Each element in p_2 corresponds with the probability that the player has the same word in the first round with the AI-agent himself. To simplify, we use a vector to represent this p_2 set, and the index, such as 0, 1, 2, etc, separately stands for the player besides our AI-agent. Thus, p_{2i} stands for one player's possibility.

The relationship between the probability of each player is undercover is established as follows. Intuitively, the probability of AI-agent is undercover is p_1. We first denote the events as follows,

A AI-agent is not undercover.
B The player's word is different from AI-agent's.
C The player's word is the same as AI-agent's.

Thus for other players, $P = P(A, B)$.

$$P(A, B) + P(A, C) = P(A) = 1 - p_1$$

Since

$$P(A, B) = 0,$$

$$P(A, C) = P(C) = p_{2i}.$$

We have

$$P(A, B) = 1 - p_1 - p_{2i}.$$

Since we can compute undercover probabilities by p_1 and p_2, from now, our task is transferred to calculate those two values. First of all, we consider those factors that may make p_1 bigger or smaller. If AI-agent is undercover, then the words put forward by other players may not be highly relevant to AI-agents word. They may have higher relevance to the other word instead. Thus we grab some other words that are relevant to AI-agents word. If AI-agent is not undercover, there must be a player get the exactly the same word in the first round with our AI-agent. If the words put forward by other players are more relevant to AI-agents word, p_1 will be set lower. On the contrary, if the words put forward by other players are more relevant to AI-agents relevant words, p_1 will be set higher. Here we give an example to further illustrate this case.

Suppose AI-agents word is queen and another word is 'king', which can be grabbed since it is relevant to 'queen'. If a word put forward by other players is 'man', which is more relevant to 'king' instead of 'queen', then AI-agents word is more likely to be the undercover word. Another case is the word put forward by other players is woman, we will get the opposite result by similar analysis.

Calculation of p_2 is more direct and intuitive. It only involves whether the word of each player is the same with AI-agents word. Whether AI-agent is undercover is not taken into consideration. Meanwhile, the only information we get is the relevance between the word put forward by each player and AI-agents word. Thus, what we have done is set a sweet interval. If the relevance between two words falls into the interval, we say those two words are the same. Otherwise, those two words are not likely to be the same.

[Sampling] In this part, after retrieving some information about other players, our purpose is to make AI-agent deliver a word according to all the information he has got so far. We now need a probability density function to sample. Here we explain how to construct such a probability density function.

Let's take a simple example: player_0 is an AI-agent, player_{1,2,3,4} are human players. During a certain round, player_{1,2,3,4} delivered [word_1, word_2, word_3, word_4]. Counting his item-in-hand in, the list would be [own_item, word_1, word_2, word_3, word_4]. Then we invoke the function *doesnt_match* to find out one outlier that is least similar to all other words in that list, say word_4 this time.

So far we got two list, one of which is [own_item, word_1, word_2, word_3] and the other is [word_4]. We invoke *most_similar()* to both of lists to extend

$$[own\_item, word\_1, word\_2, word\_3]$$

to

$$[own\_item, word\_1, word\_2, word\_3, word'\_1, ..., word'\_k]$$

and extend

$$[word\_4]$$

to

$$[word\_4, word''\_1, ..., word''\_j]$$

This operation is applied to extend the data size to better construct the following probability density function.

We define

$$mean_1 = mean([own\_item, ..., word\_3, word'\_1, ..., word'\_k])$$

$$std_1 = std([own\_item, ..., word\_3, word'\_1, ..., word'\_k])$$

$$mean_2 = mean([word\_4, word''\_1, ..., word''\_j])$$

$$std_2 = std([word\_4, word''\_1, ..., word''\_j])$$

As shown in Fig.4, the blue curve represents $Gauss(mean_1, std_1)$, denoting the possible distribution of innocent words. And the orange curve represents $Gauss(mean_2, std_2)$, denoting the possible distribution of undercover words.

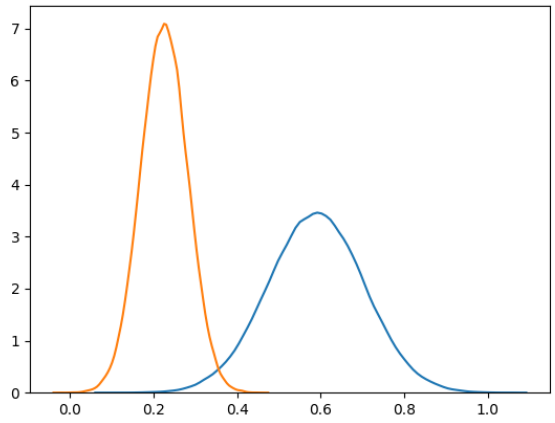

Fig. 4. coordinate transform

Recall that for AI-agent p_1 is the probability of himself being undercover and $1-p_1$ is the probability of himself being innocent. Say X denotes the random variable of words being innocent, and Y denotes the random variable of words being undercover. Then by definition above

$$X \sim Gauss(mean_1, std_1), Y \sim Gauss(mean_2, std_2)$$

Here we linearly combine X and Y into

$$Z = aX + bY$$

then

$$Z \sim Gauss(mean\_mix, std\_mix)$$

where $mean\_mix = a * mean_1 + b * mean_2$ and $std\_mix = \sqrt{(a * std_1)^2 + (b * std_2)^2}$. The distribution of Z is shown in Fig.5 (the green curve), which is used to sample.

[Transition model] In the transition part, the AI-agent updates its belief vector about the probability of each player being the undercover. And then according to the belief vector, it will vote who has the highest probability, which indicates a more likely identity as the undercover. However, if the AI-agent recognizes itself as the undercover, it cannot vote itself, so it will vote the player with the second highest probability.

We also need to record every player's vote. After all players voting in one round, the one who gets the most votes will be kicked out of the game. If the undercover is kicked out, the civilians win the game, and then the game terminates.

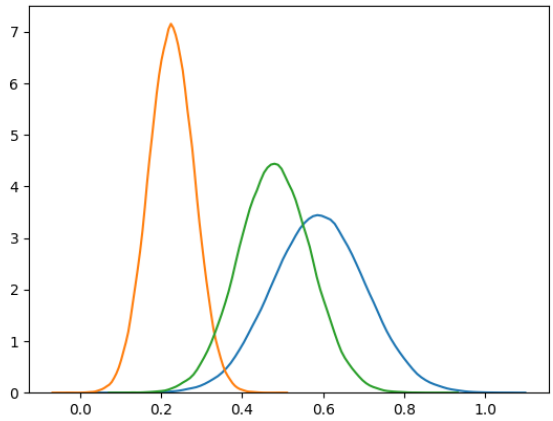

Fig. 5. coordinate transform

Otherwise, the game continues if there are still more than 2 players alive.

Then each player will be announced about who has been kicked out. Each player should remove this player out of its belief vector, then normalize it as the new belief vector in the next state(round).

V. Summary

Here we use an HMM model to predict and vote for undercover, however there are some parameters still need further adjustment. What we are thinking is to use the idea of the feature based Q-learning to automatically learn and compute the most suitable parameters. With a better combination of parameters, related to which range to sample in the gaussian distribution, the AI-agent can pick a word that may suit the situation better. Then our AI-agent can live longer in the game and win more times.

Acknowledgment

During this paper, we collaborated and discussed within a group of Fengming Zhu, Yucong Zhang, Ming Zhong and Yuyan Zhou. Great thanks to the help of Prof.Tu and those senior TA's.

References

[1] Wikipedia
[2] Gensim Tutorial