REPORT:

A Vietnamese Intelligent Dictionary Application

Table of Contents

1. Background

In modern society, being fluent in at least one foreign language has become an essential skill for citizens in many countries in the world. With the current wave of globalization in almost all countries, many people further feel the urge to learn languages to keep up with the requirements of schools, workplaces, and society. In Vietnam, many companies prioritize candidates having the ability to speak English or Chinese, and schools are also giving more opportunities for students who obtain a foreign language's certificate. Therefore, this trend is predicted to continue for a long time and will not be reversed in near future.

The popularity of learning foreign languages along with technological advances encourages the creation and development of a variety of different dictionaries. In the past, dictionaries existed in the form of printed books to help learners look up various information regarding a keyword, for example, meanings and word types. The most essential information is provided; however, it takes time to search for keywords and the limited capacity of a physical dictionary asks for some information to be left out. The introduction of electronic dictionaries now allows dictionary users to look up not only the meaning of words and phrases but also other concepts of that language: collocations, pronunciation, sentence formations, grammar, etc. These are essential for both foreign language learners and those who wish to thoroughly study their mother tongue. Other less fundamental needs of language learners are also satisfied with the large variety of non-classic dictionaries available: reverse dictionaries, thesaurus dictionaries, collocation dictionaries, etc.

However, these dictionaries still suffer from various problems such as inefficient interface, shortage of illustrative examples, and tardiness in updating new vocabulary. However, we believe the most common problem for current online dictionaries is the lack of consideration for context. Dictionaries we have today only consider the searched keyword, not its context sentence when distributing linguistics information regarding that keyword to users. Therefore, dictionary users often have to read through the whole explanation for the keyword, which can be very time consuming and inconvenient, especially when the keyword has multiple meanings and word types. To solve the problem, we hope to build an intelligent dictionary that can present meanings based on the keyword and its context sentence using some simple natural language processing methods.

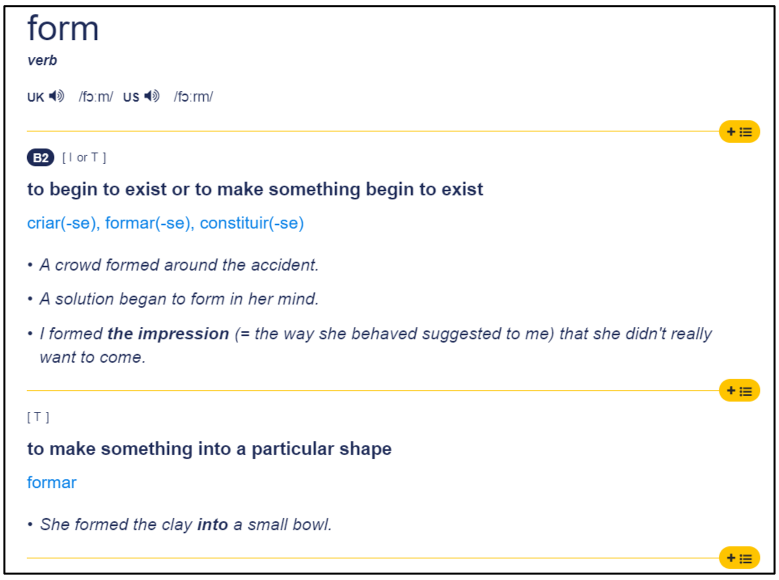
Natural language processing (NLP) has been a popular topic in recent years with various applications such as machine translation, sentiment analysis, and topic classification. Many NLP models have been proven to work efficiently and provided in readily-built libraries or toolkits, which allows not only experts but also novices to easily make use of. We realize that current online dictionaries fail to fully utilize these NLP tools to provide more advanced functions. Therefore, we wish to build an "intelligent" dictionary by applying some popular word embedding methods, for example word tokenizing tools and the Word2Vec model.

The result application is a Vietnamese – Vietnamese online dictionary designed to meet the needs of Vietnamese learners. We chose Vietnamese to build the model as we are more comfortable with the linguistics concepts of the Vietnamese language, for example word types or grammar. Moreover, there is currently a small number of NLP works focusing on Vietnamese text data compared to English, which also further encouraged us to build a dictionary in our mother tongue. Through this "intelligent" dictionary, we hope that the experience of using online dictionaries would become more convenient and satisfying for many language learners.

2. Problem

2.1. Analysis Results: The need for an "intelligent" dictionary

As previously stated, there are various things which a dictionary is capable of nowadays: searching up meanings, finding synonyms and antonyms, or suggesting related collocations. Regardless of the purpose of use, except for the reverse dictionary which predicts and gives keywords by analyzing meanings inputted by users, almost all current online dictionaries operate based on "filtering", merely giving out a long list of meanings without consideration of the context sentence.


form

verb

UK /fɔːm/ US /fɔːrm/

B2 [I or T]

to begin to exist or to make something begin to exist

criar(-se), formar(-se), constituir(-se)

- *A crowd formed around the accident.*
- *A solution began to form in her mind.*
- *I formed **the impression** (= the way she behaved suggested to me) that she didn't really want to come.*

[T]

to make something into a particular shape

formar

- *She formed the clay **into** a small bowl.*

Figure 1. Interface of the Oxford's Learner Dictionary (English – English dictionary)

Taking Oxford's Learner Dictionary as an example of classic dictionaries, we consider how it operates when users send requests to the system. When we input a keyword in the search box and hit Enter, the system would use this information to search through the database and print out filtered results for the keyword. This operation is quick and simple for the system to handle; however, the amount of time we use to find the right meaning in the context sentence can be very large. This is because the keyword may have multiple meanings or multiple word types, causing the right meaning considering the context sentence to be displayed in the latter parts of the page. Users, as a result, have to spend time scrolling through the meanings and reading the explanations and examples of multiple meanings for only one keyword.

To collect evidence which proves the need to reduce the time when using a digital dictionary, we designed a simulation test. This simulation test would estimate the average time needed to find out the correct meaning of a word in context. The estimated average time may give us a hint on whether we actually spend a large amount of time reading excessive information when searching up the right meaning for a keyword in the desired context sentence.

The simulation test includes 4 steps as follows.

Step 1: Test Samples Pick-up

In this step, we used the list of 100 pairs of keyword - context sentence in the testing dataset.

Step 2: Generate values of "meaning index"

We then search for the keywords in a Vietnamese – Vietnamese dictionary and make a table summarizing the results of the "meaning index" for each keyword. The dictionary used for this step is "Tra Từ" dictionary. We would number each meaning in the result meaning list for the keyword provided by the dictionary and find the number of the right meaning considering the context. This number is the value of the "meaning index", which illustrates the number of attempts we need to make before reaching the correct meaning if reading the meaning list from top to bottom.

Example. Vietnamese word "mua"

Table 1. Structure of meaning information: sample word "mua"

No	Word type	Meaning (Vietnamese)	Meaning (English)
1	Noun	cây bụi mọc hoang, thân và lá có nhiều lông, lá mọc đối, hoa to, màu hồng tím, quả rắn có hình trứng.	a type of wildlife tree
2	Verb	đổi tiền lấy hàng hoá, đồ vật	exchanging for goods
3	Verb	(Ít dùng) dùng tiền bạc, lợi lộc để đổi cái có lợi cho mình một cách không chính đáng	use money to gain benefits for yourself unfairly
4	Verb	bỏ nhiều công sức để rồi thu về cái không hay, ngoài ý muốn	receive unfavorable results for your efforts

When the context sentence is "Chẳng ai muốn mua cái bực vào người" (No one wants to be provoked with anger), the meaning (4) is correct, which means the "meaning index" should be set to 4. The result table for this keyword "mua" would look as below.

Table 2. Format of the simulation test result table

Keyword	Sentence	Meaning index
Mua	Chẳng ai muốn mua cái bực vào người.	4

Step 3: Evaluation

Evaluation is made by calculating the mean of column "meaning index". The smaller this number is, the fewer attempts we have to make in reading the meaning list before reaching the correct meaning of the keyword considering the context sentence. Assuming the time needed for reading each meaning is s seconds and the mean value of the "meaning index" is m, we can further make a rough estimation of the average amount of time required to find the right meaning as:

$$average\ time\ spent = m \times s\ (seconds)$$

After running the simulation test, we received the results as follows.

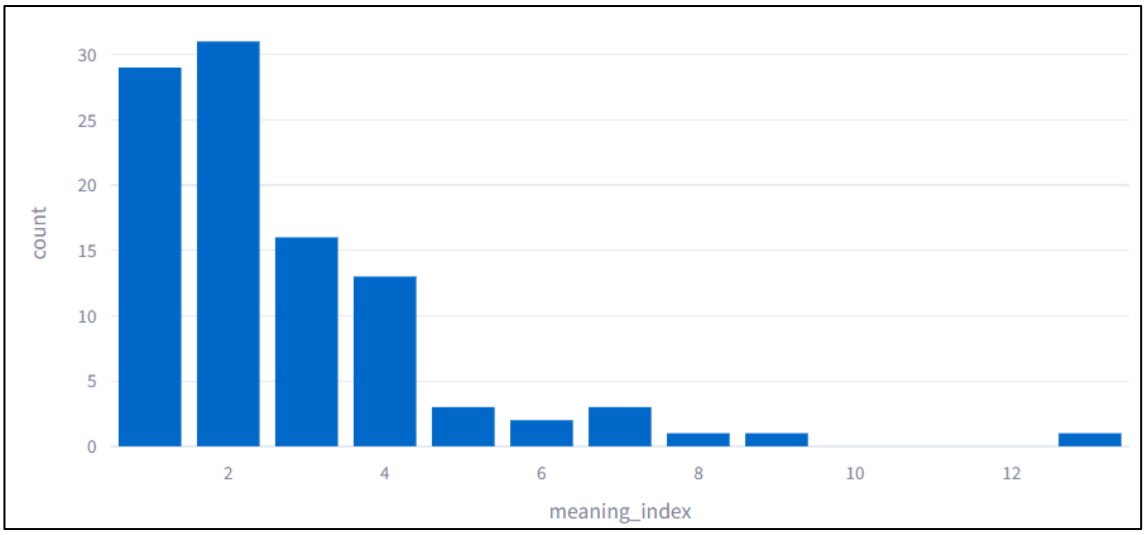


Figure 2. Histogram of 'meaning index' in simulation test

We may see that over 70% of cases, we need to read at least 2 meanings from top to bottom to find out the correct meaning in the dictionary. The average value of 'meaning index' is around 5.8; which means that on average, you need to read 5.8 meanings from the top. There also exist an extreme case when we have to read 13 meanings to reach the correct one regarding the context sentence. For these cases, the "intelligent" dictionary would be very useful and save us some minutes spent on reading unrelated information.

Therefore, we conclude that the use of our dictionary would be beneficial for many dictionary users.

2.2. Applications of Natural Language Processing

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that deals with the interaction between computers and human language, especially how computers can efficiently process and analyze text data in a large volume. NLP may be classified into many smaller subfields; however, the most popular classification for NLP subfields includes Information Extraction, Ontology, Speech Processing, and Statistical NLP. The tasks and techniques of each subfield in fact intertwine with those of another subfield, therefore, we only consider this categorization to have a general view of this technology.

With the goal to build an “intelligent” Vietnamese dictionary, we focused on some NLP tasks which are closely related to the functions of a dictionary: word segmentation (tokenization), part-of-speech tagging (POS tagging), and distributional semantics. Word segmentation and POS tagging applications are useful for analyzing the context, while distributional semantics techniques help us to quantify semantic similarities between linguistic items. In this project, we applied some techniques of distributional semantics including word2vec and word co-occurrence matrix. We believe the combination of the above NLP techniques can help improve the functions of the existing dictionaries and bring more convenience to language learners.

3. Proposal

3.1. General structure of the dictionary application

The application is constructed with 4 main components: dictionary database, text-processing model, keyword meaning's scoring model, and input-output channel.

Overall, the application would get input from users through the input-output channel. After filtering the dictionary database using inputted pairs of a keyword-context sentence, the text-processing model would process the inputted texts from users. The cleaned inputted text from the text-processing model, along with filtering results from the dictionary database, would then be used to calculate the scores by the keyword meaning’s scoring model. Finally, with outputs from the keyword meaning’s scoring model, the output channel would provide users with a list of ordered meanings of the searched keyword in the considered context.

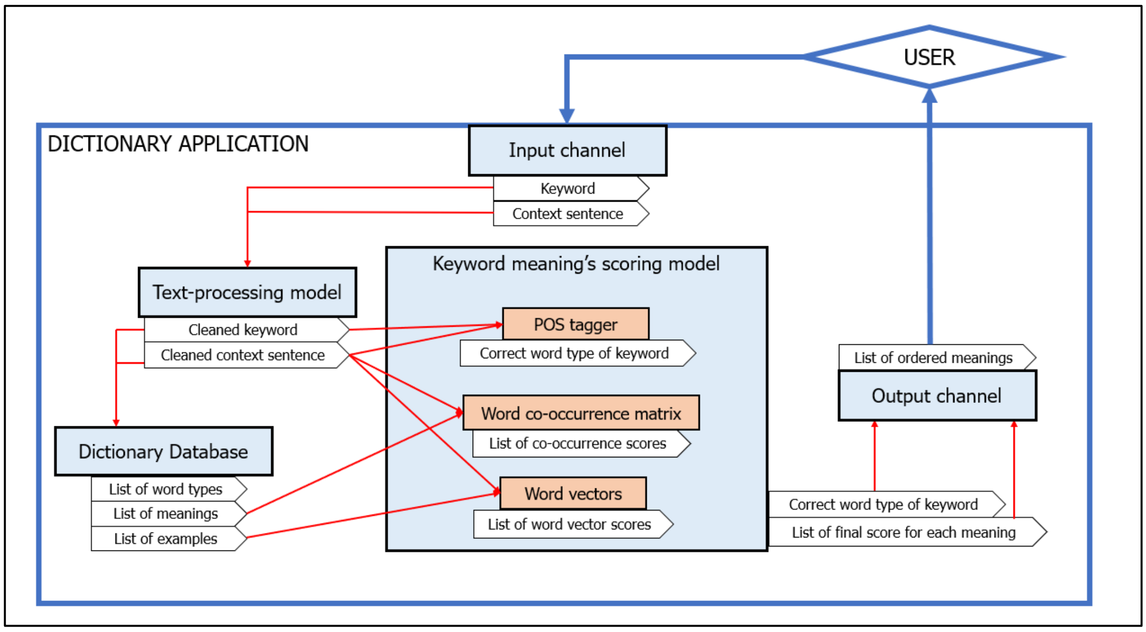


Figure 3. The proposed construction of the dictionary application

The dictionary database used in the application has the same basic structure as other current dictionary databases, but considering the memory capacity, we simplified the structure by limiting the fields to "word id", "word", "word type", "meaning" and "example". This dictionary database is scraped from the Vietnamese dictionary "Tra Từ", then cleaned and saved as CSV files into the local work directory. As the dictionary can be accessed in real-time, we merely use these CSV files in case offline use is required. As for the normal process, we would access the dictionary database of "Tra Từ" using "requests" library in Python only when inputs from users are given, update the URL with the provided keyword, and receive linguistic information accordingly.

The text-processing model is designed for two main functions: text cleaning and word tokenization. For text cleaning, the input text (including training text and testing text) would be processed through 3 following steps:

Step 1: Remove all unnecessary spaces in the raw text, especially leading and trailing whitespaces

Step 2: Get rid of unwanted characters including numbers and special characters

Step 3: Lowercase all characters in text data

After finishing the step of cleaning the raw text, we applied word tokenization to the cleaned text. We tokenized text data into a list of text at the word level including unigrams, bigrams, and multi-grams. To tokenize the text, we decided to use the tokenizer of “underthesea” library.

The keyword meaning’s scoring model includes three components: POS tagger, word co-occurrence matrix, and word vectors. The POS tagger model would tokenize the text and predict the word type of the keyword. In case the keyword exists more than once in the context sentence, the model would give results based on the first occurrence of that keyword. Using the predicted word type, we can filter the meanings scraped from the dictionary database with the corresponding word type. The second component is the word co-occurrence matrix built from text data, which is used for calculating co-occurrence scores between pairs of a meaning - context sentence. This word co-occurrence matrix is built along with the word dictionary which helps navigate the word corresponding to each index of the matrix. The last component, word vectors, generates the word vector score between pairs of context sentence – examples. We then calculate the final score as a weighted average of the co-occurrence score and word vector score, which would be used to produce outputs.

The input-output channel simply functions as an interface for users of the dictionary. The input channel would receive the keyword – context sentence pair given by users, then provide the inputted information to the other components. After the keyword meaning’s scoring model finishes calculating scores, the output channel would give users an ordered list of meanings for the searched-up pair based on the prediction results of the keyword meaning’s scoring model.

The details of how the components of our application are built would be explained in the following sections from 3.1.1 to 3.1.4.

3.1.1. POS tagger

POS tagging or part-of-speech tagging is the task of categorizing each word of a text corpus into a part of speech based on its definition and context. The POS tags provide information about the role of the keyword in that context sentence, which is often known as “word type” in linguistics.

POS tagging would be very beneficial in understanding the word given the context sentence as the definition of the keyword needs to have a word type which is fitted with its role in the sentence. Therefore, we decided to use the POS tagger model as a condition to filter out meanings of the keyword which have the correct grammatical role. Any meanings having POS tags different from the predicted POS tag should be put in the latter parts of the produced meaning list.

As we do not have a reliable training dataset for Vietnamese POS tags, we decided to use a readily-built Vietnamese NLP toolkit. The POS tagger we used for this application is built in the library "underthesea", which is a popular open-source Vietnamese NLP Toolkit. Despite some occurrence of mistakes in tokenizing and categorizing words into the correct POS tags, this POS tagger is very simple and light for executing. Moreover, the dictionary application we built is meant to deal with words belonging to main word types (nouns, verbs, adjectives, etc.); therefore, wrong POS tags in some rare cases may be safely overlooked.

The POS tagset in use contains 17 main lexical tags shown in the following table. These POS tags were originally created in a paper to cover most features of the Vietnamese language, however, they received some modifications by the developer of "underthesea". Vietnamese and English have some differences in grammar, for example the absence of tense in Vietnamese grammar, which is reflected in "Tra Từ" dictionary. Therefore, some English POS tags are grouped together to create suitable Vietnamese POS tags (herein also mentioned as "word type").

Table 3. List of POS tags used in the application

POS tag ("underthesea")	**Word type**	**Vietnamese POS tag (dictionary)**
Np	Proper Noun	Danh từ
Nc	Classifier	
Nu	Unit Noun	
N	Common Noun	
L	Determiner	
M	Numeral	
V	Verb	Động từ
A	Adjective	Tính từ
P	Pronoun	Đại từ
R	Adverb	Phụ từ
E	Preposition	Kết từ
C	Subordinating conjunction	
CC	Coordinating conjunction	

I	Interjection	Cảm từ
T	Auxiliary, modal words	Trợ từ
-y (Ny, Vy, etc.)	Abbreviation	Depends on the abbreviated word
Z	Bound morphemes	Không rõ
X	Unknown	

The purpose of the POS tagger model is to figure out the grammatical role of the keyword in a given context sentence. To do this, the POS tagger model imported from "underthesea" library would tokenize the context sentence into words labeled with their POS tags. The results of the tagger model would look as below, from which we can get the POS tag of the keyword. The model would consider the first occurrence of the keyword in the sentence if the keyword exists more than once.

Example. Results of the POS tagger model from "underthesea" library for context sentence "Chàng trai 9X Quảng Trị khởi nghiệp từ nấm sò."

[('Chàng', 'Nc'), ('trai', 'N'), ('9X', 'N'), ('Quảng Trị', 'Np'), ('khởi nghiệp', 'V'), ('từ', 'E'), ('nấm', 'N')]

The results of the tagger model may fall into the following three cases:

Table 4. How the application proceeds based on results of the POS tagger model

Case No.	Keyword	How to proceed
1	Same as a tokenized word	Get the POS tag of the corresponding tokenized word
2	Included in a tokenized word	Apply POS tagger model to the tokenized word
3	Include 2 or more tokenized words	Use the POS tag of the first tokenized word included

3.1.2. *Word co-occurrence matrix*

Word co-occurrence matrix is a matrix showing the number of times with which two words co-occur in a set of training text samples within a window. The window size can be set arbitrarily by developers to examine the text data in different scales for different purposes. Having a word co-occurrence matrix, we can get a rough estimation of how often two words are seen together in a sentence or proximity with a fixed size.

Table 5. Sample of a word co-occurrence matrix

	Word 1	**Word 2**	**…**	**Word n**
Word 1	0	2	…	8
Word 2	4	1	…	0
…	…	…	…	…
Word n	3	5	…	0

In the application, we utilized the word co-occurrence matrix to compute a co-occurrence score between the context sentence and the meanings provided by the dictionary database. Believing that the context sentence should contain words that often co-occur with the words in the keyword's definition, we compute the co-occurrence scores for each context sentence – meaning pair using the formula defined in 4.1.5 (a). The smaller the co-occurrence score is, the higher chance the context sentence - meaning pair to be the correct pair. In other words, we would choose the meaning belonging to the pair having the lowest co-occurrence score.

To build the word co-occurrence matrix from scratch, we need some text data to train the matrix. The train text data was pre-processed in the text-processing model and passed to the training function. The function started with creating the word dictionary by assigning each word to an index. Using numpy.array in Python, we then created a zero matrix whose shape is defined as the size of the word dictionary. Next, the training function would loop through all sentences (a sentence is a list of cleaned words) in the train text data, and update the word co-occurrence matrix by adding 1 to the corresponding value of the word pair in the matrix whenever these 2 words co-occur within the specified window size.

However, this approach suffers from memory problems as our computer's RAM does not have enough memory to store large-size NumPy matrices. To curb the issue, we applied the 4 following measures to reduce the memory needed when building the word co-occurrence matrix.

Measure 1: Build matrix with int16 type

Numpy matrices in Python have the default data type of "float". This data type takes much more memory compared to int16 (16-bit), and this excessive memory use is not needed as we deal with only integers in the word co-occurrence matrix. However, this data type sets a maximum limit to the value of each element. Therefore, in the function, elements of the matrix would not be updated if their values reach 32767. In this small train text dataset, we believe that the majority of word pairs would not co-occur more than this limit, ensuring that some insignificant miscounts would not considerably affect the scoring model's performance.

Measure 2: Split training text data into small batches and update the matrix after training each batch

We divided the train text data into batches, each batch containing around 200,000 sentences. The results of the word co-occurrence matrix and word dictionary of the previous batch would be used as the input for the next batch to reduce the amount of memory needed for training.

Measure 3: Select an optimal small window size

We decided to count the number of times two words occur in the corpus in a window size of 8. Here is an example of how the looping works:

Example. Consider this sentence "I said hi to her but she said nothing back".

We assume that the word dictionary looks as below.

Table 6. Sample table of a word dictionary (first 10 words)

Word	I	said	hi	To	her	But	She	said	nothing	back
Word ID	1	2	3	4	5	6	7	8	9	10

Then, a sample of how the word co-occurrence matrix M would be updated is shown below.

Table 7. Illustration of how co-occurrence matrix is built (first 3 steps)

	Words considered to update matrix	**Updated elements**
Step 1	["I", "said", "hi", "to", "her", "but", "she", "said", "nothing"]	M[1][n]+=1 for n∈[2,9]
Step 2	["said", "hi", "to", "her", "but", "she", "said", "nothing", "back"]	M[2][n]+=1 for n∈[3,10]
Step 3	["hi", "to", "her", "but", "she", "said", "nothing", "back"]	M[3][n]+=1 for n∈[4,10]

This word co-occurrence matrix, therefore, would not be a symmetric matrix as a result of the usual approach. However, this approach helps reduce the number of calculations and loops needed, and less memory would be used as a result.

Measure 4: Set a minimum threshold to filter out rarely observed words

This approach is used to remove words that are not used frequently, which helps reduce memory needed and avoids unwanted misleading text such as misspelled words. In this model, we set the minimum required frequency of a word in each batch to 30 times.

3.1.3. *Build word vectors with Word2Vec*

We applied the CBOW architecture to train the word vectors. Functions from "gensim" library were utilized to train the cleaned text data which has been readily processed by the text-processing model. The trained Word2Vec model would be saved to calculate the word vector score between the context sentence inputted by users and the example sentence which is provided by the dictionary database.

3.1.4. *Scoring algorithm*

The scoring algorithm we used in the keyword meaning's scoring model includes three formulas defined as follows.

a. Co-occurrence score (z_1)

Assume we have two sentences (or phrases) s_1 and s_2 having lengths larger than 0, and the number of times which two words $w_1(w_1 \in s_1)$ and $w_2(w_2 \in s_2)$ co-occur is denoted as $f(w_1, w_2)$. A closer semantic relationship between two sentences may be concluded if we receive a higher value calculated by the formula below:

$$\frac{\sum_{w1 \in s1} \sum_{w2 \in s2} f(w_1, w_2)}{length(s_1) \times length(s_2)}$$

In this model, assuming that the word co-occurrence matrix we built is M and i_1, i_2 are the indexes of w_1, w_2 in the matrix respectively, we can calculate $f(w_1, w_2)$ as:

$$f(w_1, w_2) = \frac{M[i_1][i_2] + M[i_2][i_1]}{2}$$

However, in real practice, the word co-occurrence we built would not be able to cover all Vietnamese words, and we cannot obtain the number of times two words w_1, w_2 co-occur if at least one of them is not included in the built word dictionary. Therefore, we would calculate the number of valid pairs of (w_1,w_2) (both w_1,w_2 are included in the word dictionary) as $cnt(f)$, and the sum of $f(w_1, w_2)$ for all valid pairs as $sum(f)$. Considering the word vector score, we also modified the formula above so that a smaller value would be returned if two sentences have a closer semantic relationship. In the application, we computed the co-occurrence score between the context sentence (s_c) and the meaning (s_m), and the final formula would look as follows.

$$z_1(s_c, s_m) = \begin{cases} Null\ if\ cnt(f) = 0 \\ 10\ if\ cnt(f) \neq 0, sum(f) = 0 \\ \dfrac{cnt(f)}{sum(f)}\ if\ cnt(f) \neq 0, sum(f) \neq 0 \end{cases}$$

If $cnt = 0$, no valid pairs of (w_1, w_2) are observed; therefore, the co-occurrence score cannot be calculated, and null value would be given as the result.

If $cnt(f) \neq 0, sum(f) = 0$, all pairs of (w_1, w_2) do not co-occur in the trained text corpus; therefore, z_1 should be given a maximized value to indicate that two sentences have little relationship with each other. We estimated this maximized value to be 10 considering that there is almost zero possibility that the sum of $f(w_1, w_2)$ for 10 pairs is smaller or equal to 1.

Looking at the formula, we conclude that a smaller value of z_1 suggests more similarity or a closer relationship observed between two sentences.

b. Word vector score (z_2)

Similarly, assuming we have two sentences (or phrases) s_1 and s_2, if the distance between 2 word vectors $w_1 (w_1 \in s_1)$ and $w_2 (w_2 \in s_2)$ is denoted as $d(w_1, w_2)$, the word vector score in theory is calculated as defined by the formula below. A smaller value received from this formula may indicate that two sentences are more semantically similar.

$$\frac{\sum_{w_1 \in s1} \sum_{w2 \in s2} d(w_1, w_2)}{length(s_1) \times length(s_2)}$$

In the application, we use the Euclidian distance to calculate the distance $d(w_1, w_2)$ between word vectors. As we want to calculate the similarity score for two sentences with different role (context sentence s_c and example sentence of keyword s_e), we proposed this improved version of the original formula:

$$\frac{\sum_{w_1 \in s_c} min_{w_2 \in s_e}(d(w_1, w_2))}{length(s_c)}$$

This formula would calculate the average distance of all pairs of (w_1, w_2) with smallest distance for each $w_1 \in s_c$. This formula still needs further improvement since similar to z_1, there might exist words which are not included in the set of word vectors built by the application. We would fail to calculate the distance between w_1 and w_2 in this case, therefore, we proposed an alternative formula for z_2:

$$z_2(s_c, s_e) = \begin{cases} Null\ if\ cnt = 0 \\ \dfrac{sum(d)}{cnt(d)}\ if\ cnt \neq 0 \end{cases}$$

In this formula, $cnt(d)$ is number of valid pairs of (w_1,w_2) (both w_1, w_2 have corresponding word vectors), and $sum(d)$ is the sum of $d(w_1, w_2)$ for all valid pairs (valid pair is the pair with smallest distance $d(w_1, w_2)$ for each w_1).

c. Final score (z)

We wish to calculate z as a weighted average of two scores z_1 and z_2. Give weight c_n to score z_n, we have the following formula to calculate the final score of each meaning s_m (meaning sentence or phrase) having example sentence s_e (s_m, s_e are provided by the dictionary) considering the context sentence s_c provided by a user:

$$z(s_c, s_e, s_m) = \frac{z_1(s_c, s_m) \times c_1 + z_2(s_c, s_e) \times c_2}{c_1 + c_2}$$

In case the meaning s_m have multiple examples, we choose s_e with smallest value of $z_2(s_c, s_e)$ from the available list of example sentences of meaning s_m.

However, as z_1 and z_2 may have null values as mentioned in 4.1.5(a) and 4.1.5(b), we would only use the formula above in case both z_1, z_2 are not null. For the other cases, we calculate z as:

$$z(s_c, s_e, s_m) = \begin{cases} \dfrac{z_1(s_c, s_m) \times c_1 + c_2}{c_1 + c_2}\ if only\ z_2(s_c, s_e)\ is\ Null \\ \dfrac{c_1 + z_2(s_c, s_e) \times c_2}{c_1 + c_2}\ if only\ z_1(s_c, s_m)\ is\ Null \\ Null\ if\ both\ z_1(s_c, s_m),\ z_2(s_c, s_e)\ is\ Null \end{cases}$$

The reason why we assign 1 to the value of the co-occurrence score and the word vector score is an intuition that meanings or example sentences consisting of words not included in the trained word co-occurrence matrix and word vectors would be used less frequently. Therefore, we would give a maximized value to z_n with null values to indicate that this meaning is not commonly used. With formulas defined above, z_1, z_2 should have values smaller than or equal to 1, and list of scores z_1, z_2 for all meanings considered would be normalized as well. As a result, we used 1 as the alternative value for z_1, z_2 to calculate z if z_1 or z_2 is null.

A smaller final score means that the considered meaning has a higher probability to be the correct definition of the keyword regarding the context sentence.

Through multiple trials, we decided that the weight combination (1, 2) works the best for our application.

4. Implementation

4.1. Baseline

All the main processes including scraping text data and dictionary data; training word dictionary, word co-occurrence matrix, and word vectors; building keyword's meaning scoring model are written in Python code and run on Jupyter Notebook. Excel is also used to make some modifications which require human judgment. For example, we used Excel to the extracted data in the simulation test (adding "meaning index" to each keyword), and to the test data (adding the correct meaning of each randomly chosen pair of a keyword – context sentence). To build the interface, we used Streamlit and deployed the built app with Streamlit Community Cloud.

4.2. Hyperparameter Settings

The table below summarized the hyperparameter settings we used to build the application. These parameters were chosen after doing a random search based on the results of the testing dataset. We also consider the memory limit and processing ability that we possess when using these hyperparameters.

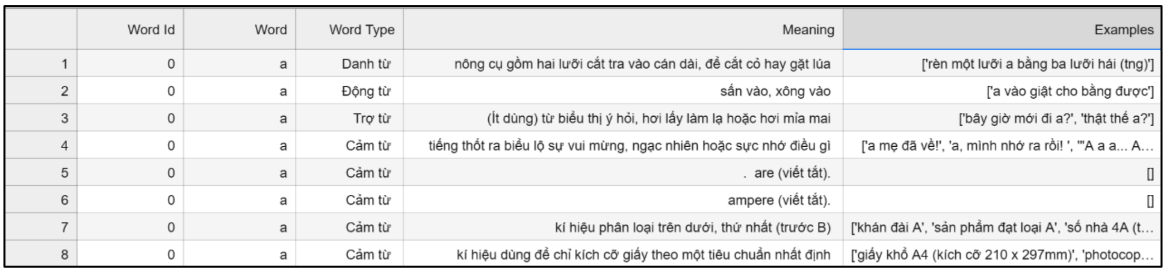
Table 8. Specifications for hyperparameters in each function

Application Component	Function	Hyperparameter	Value
Text processing model	Text cleaning	min_sentence_length	8
The keyword meaning's scoring model	Word co-occurrence matrix	window_size	8
		min_count_words	30
	Word2Vec	min_count	100
		vector_size	100
		window	8
		max_vocab_size	50000

4.3. Datasets

To build the application, the use of two main datasets was required: a dictionary database (for searching for meanings and examples of each keyword) and raw text data (for building word vectors and word co-occurrence matrix).

The dictionary database in the application was scraped from the Vietnamese – Vietnamese dictionary "Tra Từ". We scraped the data using Python with library "requests", and parsed LXML with library "bs4". A sample of the scraped dictionary data is shown in the following figure:



	Word Id	Word	Word Type	Meaning	Examples
1	0	a	Danh từ	nông cụ gồm hai lưỡi cắt tra vào cán dài, để cắt cỏ hay gặt lúa	['rèn một lưỡi a bằng ba lưỡi hái (tng)']
2	0	a	Động từ	sấn vào, xông vào	['a vào giật cho bằng được']
3	0	a	Trợ từ	(Ít dùng) từ biểu thị ý hỏi, hơi lấy làm lạ hoặc hơi mỉa mai	['bây giờ mới đi a?', 'thật thế a?']
4	0	a	Cảm từ	tiếng thốt ra biểu lộ sự vui mừng, ngạc nhiên hoặc sực nhớ điều gì	['a mẹ đã về!', 'a, mình nhớ ra rồi! ', '"A a a... A…
5	0	a	Cảm từ	. are (viết tắt).	[]
6	0	a	Cảm từ	ampere (viết tắt).	[]
7	0	a	Cảm từ	kí hiệu phân loại trên dưới, thứ nhất (trước B)	['khán đài A', 'sản phẩm đạt loại A', 'số nhà 4A (t…
8	0	a	Cảm từ	kí hiệu dùng để chỉ kích cỡ giấy theo một tiêu chuẩn nhất định	['giấy khổ A4 (kích cỡ 210 x 297mm)', 'photocop…

Figure 4. Sample of scraped dictionary data

The training text datasets we used to build word vectors and word co-occurrence matrix for this application are summarized in the table below. As we only use the text for training, the only column being utilized was the one which contains raw Vietnamese text data.

Table 9. Datasets used in the project

No	Dataset Name	Details	Source
1	"News Dataset Vietnamese"	290,282 raw Vietnamese articles crawled from "Bao Lao Dong"	kaggle.com/datasets/phamtheds/news-dataset-vietnameses
2	Scraped Wikipedia Text	Scraped text from articles of segment "bai viet tot" (high-quality articles), "bai viet chon loc" (selected articles), and top articles	vi.wikipedia.org (self-scraped)
3	"Vietnamese wikipedia corpus 2020"	Cleaned and processed text data from Wikipedia articles written in Vietnamese in year 2020	kaggle.com/datasets/89a41b1ac1ff10900b27fdd85da6528fd82b5f72e5b6c45d02213d034243fd8e

5. Results

5.1. Metrics for evaluation

Since there are no available testing methods to evaluate the performance of the application, the testing data had to be created from scratch. For this evaluation purpose, I used the accuracy of predicting keyword meaning as the main metric. The total execution time of the dictionary application (text processing, filtering, score calculation, and final output generation) for each pair of a keyword – context sentence should also be evaluated.

The accuracy metric is calculated through the following steps.

Step 1: Generate output which is an ordered list of the meanings, then extract the first meaning from that resultant list

Step 2: Compare the first meaning with the correct meaning of the keyword regarding its context

Step 3: If the two meanings are the same, the prediction is counted as "correct"

Step 4: Count the total number of "correct" predictions among all testing samples

Another metric we want to use is the prediction score. Prediction score is defined as the average of the "meaning index" for the correct meaning considering the ordered list of meanings produced by the application. This score would try to estimate the average number of efforts we need to make to reach the correct meaning using this application. The ratio score is similar to the prediction score; however, it also takes the number of meanings for the keyword into consideration. Detailed explanations of the evaluation metrics are shown in the table below.

Table 12. Metrics for evaluation

Metric	Explanation
Accuracy (%)	Meaning prediction accuracy over all pairs
Absolute accuracy (%)	Meaning prediction accuracy over pairs with correct prediction of POS tags
Prediction score	Average of "meaning index" values for correct meaning from resultant ordered list of meanings over all pairs
Ratio score	Average of ratio values (= "meaning index" of the correct meaning / length of resultant ordered list of meanings) over all pairs

Time (s)	Average execution time of score calculation for each pair

The goal of our application, therefore, is firstly to maximize the accuracy and the absolute accuracy, two metrics representing how well the application is predicting the correct meaning of the searched keyword based on its context. The value of the last three metrics, however, should be reduced as much as possible. A smaller value of the prediction score and the ratio score shows that the application is helpful in reducing the attempts which dictionary users have to make in finding the correct meaning when searching up keywords. As time is also an important factor that is considered by users, we also wish to minimize the execution time of the “intelligent” dictionary.

5.2. Evaluation Results

As there is no training dataset available, we created a set of test data that contains n randomly chosen keyword – context sentence pairs (n=100). These pairs of keyword – context sentence were generated from the “Vietnamese wikipedia corpus 2020” dataset. For each pair of a keyword – context sentence, we assigned the correct meaning according to the “Tra Từ” dictionary by hand. We would then run the model on these n testing samples to check the results of each metric.

The results are summarized as follows.

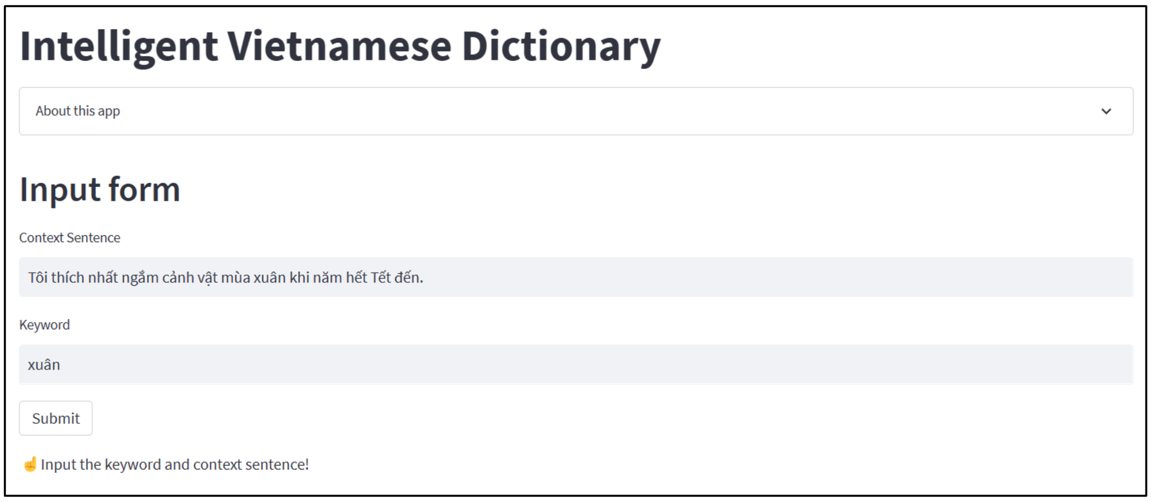
Table 13. Evaluation Results (number of testing samples=100)

Accuracy	**Absolute accuracy**	**Prediction score**	**Ratio score**	**Time**
47	**60.256**	**2.48**	**0.472**	0.053

The accuracy and absolute accuracy are quite low for all models (47% and 60.26%); however, as the model got a low prediction score, the correct meaning was ranked in the top of the predicted list of meanings. On average, users only need to read 2.48 meanings to find out the correct meaning of the keyword in provided context. Moreover, the average amount of time taken for a prediction to be made is 0.05s, which is good enough for users as they would not feel any long delay while waiting for the result.

5.3. Final dictionary application

We built the final dictionary application using Streamlit, and the application is free for anyone to access and evaluate how it works. The dictionary application simply includes two main parts: an input form and the resultant table of meanings for inputted information. The dictionary receives the pair of a keyword - context sentence inputted by the user, then shows the ordered list of meanings only after around 1~2 seconds. The interface of our dictionary application looks as follows.



Intelligent Vietnamese Dictionary

About this app

Input form

Context Sentence: Tôi thích nhất ngắm cảnh vật mùa xuân khi năm hết Tết đến.

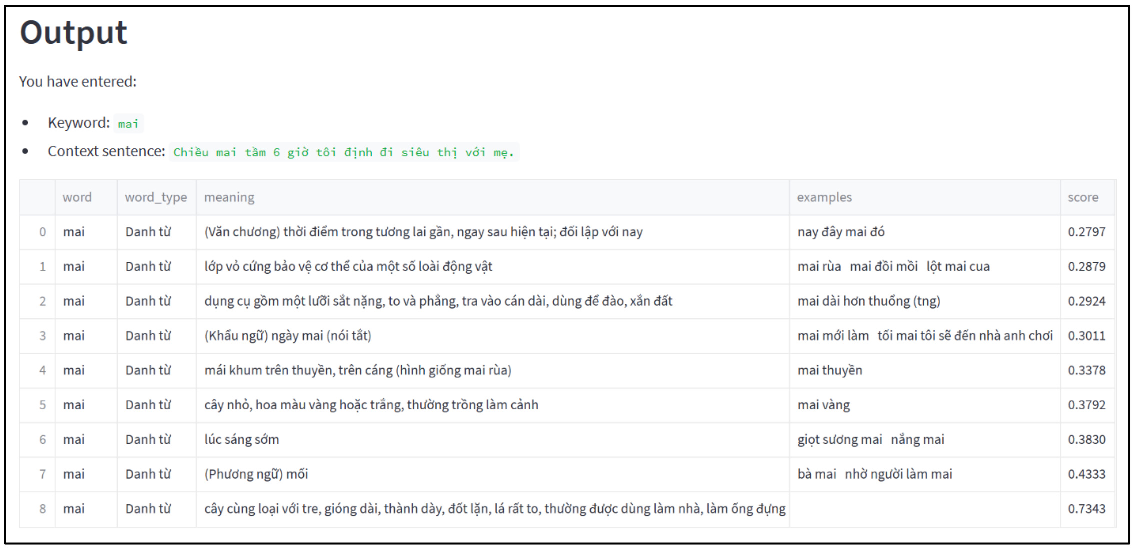
Keyword: xuân

Submit

☝ Input the keyword and context sentence!

Figure 5. The input channel of the dictionary application

After users input the context sentence and keyword into the "Input form" and hit "Submit" inside the form, the application would start scraping the "Tra Từ" dictionary and calculating the scores for the list of meanings. An example of running results that the application produced is shown below.



Output

You have entered:

- Keyword: `mai`
- Context sentence: `Chiều mai tầm 6 giờ tôi định đi siêu thị với mẹ.`

	word	word_type	meaning	examples	score
0	mai	Danh từ	(Văn chương) thời điểm trong tương lai gần, ngay sau hiện tại; đối lập với nay	nay đây mai đó	0.2797
1	mai	Danh từ	lớp vỏ cứng bảo vệ cơ thể của một số loài động vật	mai rùa mai đồi mồi lột mai cua	0.2879
2	mai	Danh từ	dụng cụ gồm một lưỡi sắt nặng, to và phẳng, tra vào cán dài, dùng để đào, xắn đất	mai dài hơn thuổng (tng)	0.2924
3	mai	Danh từ	(Khẩu ngữ) ngày mai (nói tắt)	mai mới làm tối mai tôi sẽ đến nhà anh chơi	0.3011
4	mai	Danh từ	mái khum trên thuyền, trên cáng (hình giống mai rùa)	mai thuyền	0.3378
5	mai	Danh từ	cây nhỏ, hoa màu vàng hoặc trắng, thường trồng làm cảnh	mai vàng	0.3792
6	mai	Danh từ	lúc sáng sớm	giọt sương mai nắng mai	0.3830
7	mai	Danh từ	(Phương ngữ) mối	bà mai nhờ người làm mai	0.4333
8	mai	Danh từ	cây cùng loại với tre, gióng dài, thành dày, đốt lặn, lá rất to, thường được dùng làm nhà, làm ống đựng		0.7343

Figure 6. Example of the dictionary application's output

The application we built may be accessed using the link below:

https://vndictapp-ek4r3xdf5irkysabnqhewt.streamlit.app/

5.4. *Explanations of results*

The low accuracy score of the application may be attributed to various reasons, one of which is the low accuracy of the POS tagger model from "underthesea". This model only managed to predict the POS tag of the keyword correctly in 78 out of 100 testing samples, heavily affecting the performance of our application. False predictions of this POS tagger model often came from sentences with confusing or complex grammatical structures, for example ones in which the keyword is included in a noun clause, or when the keyword is an abbreviated form of a longer word. The POS tagger model would be easily tricked in these cases, causing the dictionary application to sort the final ordered list of meanings wrongly (as meanings having the same POS tag with the predicted POS tag would be put on the top). The inconsistency between the system of POS tags used by the POS tagger model of "underthesea" library and the word types in the dictionary database can be another factor affecting the performance of our application.

The second main problem lies in the dictionary database which we scraped from "Tra Từ" dictionary. For some examples of the keyword – context sentence pair, we observed that the meaning list provided by the dictionary did not have the correct meaning for that keyword considering the context. This is especially obvious when the keyword is used in informal conversations, or in dialogues between young people on social media.

Example. Keyword "quẩy" in context sentence "Ngày mai tụi mình đi quẩy phố đi bộ Hà Nội nhé."

Correct meaning: go out to have fun with other people.

List of meanings provided by the dictionary:

1/ [N] món ăn làm bằng bột mì vắt thành thỏi dài, rán phồng (a kind of fried food)

2/ [V] mang đi bằng quang gánh (to bring something using a carrying pole)

3/ [V] (Ít dùng) mang đi bằng cách móc vào một đầu đòn đặt trên vai (to carry something by putting it on a pole located in your shoulder)

When none of the provided meanings from the dictionary is correct, there is no possibility that the application may give users the correct meaning of the keyword. In some other rare cases, the information carried by the definition is correct but the word type for this definition is wrong, causing our application to fail at catching the correct meaning. Another problem is the lack of example sentences in the dictionary database. Without the example sentences, the application cannot calculate the word vector score whose weight is twice the co-occurrence score's weight in the calculation of the final score, considerably reducing the accuracy of our application.

Apart from those external factors, the limitations in our training process also have some negative impacts on the performance of the final application. With the limited RAM resource of our computer, the amount of raw text data we may train is small compared to other readily-built NLP models. The process of training the word co-occurrence matrix also suffers from degradation of accuracy due to the measures we utilized to make up for the lack of memory resources. Therefore, the word co-occurrence matrix and word vectors we trained may fail to catch a lot of patterns in Vietnamese words and grammar, lowering the accuracy and absolute accuracy of the final dictionary application.

Lastly, as we have previously stated in 5.2, the results of these tests should only be used as a reference, not as evidence to draw any definitive conclusion. The absence of suitable testing datasets required us to conduct human judgment tests which are time-consuming, cutting down the size of the test dataset we built as a result. Therefore, instead of reaching any decisive conclusion about the performance evaluation of our dictionary application, we conclude that its performance needs to be further analyzed through feedback of more dictionary users in real practice.

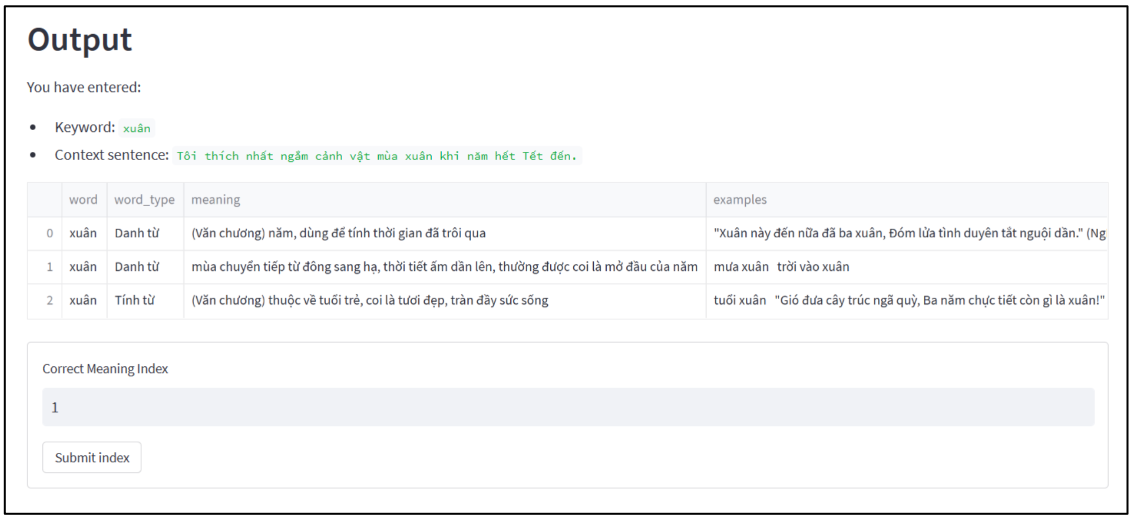
5.5. Potential improvements

Looking at the limitations in 5.4, we realized that various improvements may be conducted on the final dictionary application we built to enhance its performance. These improvements can be categorized into three groups: improving external factors, transforming the model, and utilizing users' feedback.

The external factors, as previously summarized, regard the resources we used to build the intelligent dictionary: raw text data, dictionary database, and POS tagger model. As the amount of trained raw text data and the resulting word co-occurrence matrix's accuracy is dependent on the available time, memory, and computing resources, we may try to train data on Cloud or other services to increase the size of resources. Improvements in the Vietnamese dictionary database and POS tagger model, however, are difficult to be achieved in a short time as they depend on the efforts of the developers. Therefore, collaborations with developers of these NLP resources are required to improve the performance of their products and our dictionary application.

Transforming the model, on the other hand, considers the structure and mathematics algorithm which constructs our model. For example, the scoring algorithm may be further optimized to produce a final score which better reflects the relationship between the context sentence and a meaning phrase, and between the context sentence and an example sentence. This improvement requires additional time and effort to conduct various experiments to evaluate the performance of different approaches, which can be a topic for other research in the future.

The last potential improvement we proposed is to utilize users' feedback to create a self-improvement mechanism for the application. This improvement might be achieved by adding a "feedback" form to the interface of our dictionary application. When the application showed the ordered list of meanings, users would be required to rate whether the application has made a correct prediction or not by inputting the index of the correct meaning. If the correct meaning is provided in the first position of the resulted list, users would enter "0", and the correct meaning's index in case the dictionary's prediction is wrong. Using this feedback, the application may add new examples with the correct meaning provided by users to the database, which would improve its prediction when a similar context sentence is inputted in the future.



Output

You have entered:

- Keyword: `xuân`
- Context sentence: `Tôi thích nhất ngắm cảnh vật mùa xuân khi năm hết Tết đến.`

	word	word_type	meaning	examples
0	xuân	Danh từ	(Văn chương) năm, dùng để tính thời gian đã trôi qua	"Xuân này đến nữa đã ba xuân, Đóm lửa tình duyên tắt nguội dần." (Ng
1	xuân	Danh từ	mùa chuyển tiếp từ đông sang hạ, thời tiết ấm dần lên, thường được coi là mở đầu của năm	mưa xuân trời vào xuân
2	xuân	Tính từ	(Văn chương) thuộc về tuổi trẻ, coi là tươi đẹp, tràn đầy sức sống	tuổi xuân "Gió đưa cây trúc ngã quỳ, Ba năm chực tiết còn gì là xuân!"

Correct Meaning Index

1

Figure 12. Sample interface of the dictionary application with "feedback" functions

For this approach, we need to store the dictionary database in a cloud or physical computer which allows modifications, not scraping the Vietnamese dictionary directly from the Internet when calculating scores as the current scoring function. This improvement, however, fails to deal with the wrong classification of POS tags if we continue to use the built POS tagger model from “underthesea” without any modifications. Therefore, the application may observe an increase in the “absolute accuracy” metric, but not necessarily its “accuracy”.

6. Conclusion

Through a simulation test and individual interviews with some Vietnamese speakers, we realized the need for an “intelligent” dictionary that provides the meaning of words by analyzing their context. Various NLP techniques such as Word2Vec and POS tagger proved to be useful in building this dictionary as they can provide information on the semantic and grammatical structure of text. We have built some models with different techniques and hyperparameters to compare the results, and created the final application using the most accurate and efficient model based on the prediction results of the test dataset. The final dictionary application we built has shown potential to improve the user experience of current traditional dictionaries: less time is required to read explanations for meanings, and a final score is given as a reference for users to analyze the role of a keyword in the context. However, various limitations still remain, especially those regarding the dictionary’s accuracy in predicting the correct meaning of a keyword and its training time for raw text data. These problems may be properly addressed through some improvements including efforts in enhancing available NLP resources, transforming the scoring algorithm, and proposing a self-improvement mechanism through users’ feedback. The application’s feedback function is being built, however, we ask for more time and effort to test out its efficiency. To improve the application’s accuracy, we also planned to further optimize our scoring algorithm and the current method we used to build the word co-occurrence matrix. Through these measures, we hope the application can reach a consistently high accuracy, allowing our proposed model to be efficiently utilized in other currently popular applications, for example dictionary extensions on browsers or Kindle. We believe the potential for our application in the field of linguistics and education in the future is enormous, therefore, more efforts should be spent on research focusing on this task.

Acknowledgement

This paper could not be written without the support of many people.

Many thanks to my current workplace who gave me the chance to join the Udacity course and get to learn many things about the work of a data scientist.

I am also thankful for my two friends, Hoai Anh and Giang, for giving me great encouragement in writing this paper. They also participated as experimental users to give many ideas on how I should improve the application, which is reflected on the section "Potential improvements".

Lastly, many thanks to my dearest family who has been supporting me in all my decisions through these 4 years of studying abroad. They are my greatest source of happiness and support for my whole life. To my family, I give everything, including this.