Text mining by using Python 2024 Report

Worrapat Cheng

Nanthipat Kongborrirak

Pontakorn Yuttakit

Abstract

In this report, we present our work as a team from Thailand participating in Professor Kazuyuki Motohashi's "Text-mining by using Python" project. The project focuses on implementing a dual attention machine learning model and a technology-market concordance matrix to analyse companies' patent and product data, with the goal of suggesting potential product opportunities in the market. We detail our approach to web crawling, topic modelling, keywords extraction using dual attention model, and concordance matrix. Our findings contribute to the implementation of real-world data in Thai data.

Introduction

This internship is a part of Professor Kazuyuki Motohashi's "Text-mining by using Python" project, a project focused on the implementation of dual attention machine learning model and technology-market concordance matrix on companies patent and product data to result in suggestions of possible product opportunities in the market. The project aims to provide valuable insights into potential product development areas by analysing and extracting information from patents and product data using natural language processing techniques. The dual attention model is employed to identify relevant web pages containing product details and extract product-related keywords, while the technology-market concordance matrix helps transform technology information into market information, revealing potential market opportunities for specific technologies.

Web crawling

The entire project requires data on two fields; technology data represented by each company's patents and market data represented by each company's products. As the patent datasets were provided by Professor Kazuyuki Motohashi, the product datasets were not. Therefore, before we could proceed with model training and technology opportunity suggestion, the product data must first be retrieved. In this process, text mining techniques, web crawling and web scraping, were implemented in order to retrieve html content of company websites from which we can later apply dual-attention model to extract product keywords that will then represent the "market" data of each company.

The total number of company urls provided was 77,071 urls, 40,757 were scraped in the year prior to that of this report. Thus web crawling and web scraping techniques were implemented on a total of 36,314 urls from 77,071 urls. As the provided urls did not consist of url attributes equally, they were normalised to only their scheme and domain using **parse.urlparse()** function for urllib library.

```
from urllib.parse import urlparse

url = "www.example.com/path/to/resource"
parsed_url = urlparse('https://' + url)

print(parsed_url) # Output: https://www.example.com
```

Moreover, in order to parse and retrieve html content of each page's response, the **response** and **bs4**(BeautifulSoup) libraries were implemented.

```
!pip install beautifulsoup4
```

```
from bs4 import BeautifulSoup

# html_content is a string of html content requested from a web page
soup = BeautifulSoup(html_content, "html.parser")
```

Note that some pages contain javascripts, a framework which could limit the capability of using response and BeautifulSoup altogether. To avoid overlooking hyperlink that could be of importance (i.e., contain product-related keywords), **playwright** (Microsoft, 2020) was used as an alternative to scrape data from the webpage.

```
!pip install playwright
!playwright install
!playwright install-deps
```

```python
from playwright.async_api import async_playwright
import urllib.parse
from urllib.parse import urlparse

async def get_javascript_html(browser, url_parse_result:
urllib.parse.ParseResult) -> str:
    """
    Fetches the full HTML content of a URL, including JavaScript
rendering, using Playwright.

    Args:
        browser (playwright.async_api.Browser): An open Playwright
browser instance.
        url_parse_result (urllib.parse.ParseResult): The parsed URL
object representing the target URL.

    Returns:
        str: The complete HTML content of the page after JavaScript
execution and scrolling.
    """
    try:
        async with await browser.new_context(user_agent='Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/125.0.0.0 Safari/537.36') as context:
            page = await context.new_page()
            await page.goto(url_parse_result.geturl(), timeout=60000)

            # Scroll to the bottom of the page
            _prev_height = -1
            _max_scrolls = 100
            _scroll_count = 0

            while _scroll_count < _max_scrolls:
                await page.evaluate("window.scrollTo(0,
document.body.scrollHeight)")
                await page.wait_for_timeout(500)
                new_height = await
page.evaluate("document.body.scrollHeight")
                if new_height == _prev_height:
                    break
                _prev_height = new_height
                _scroll_count += 1

            html_content = await page.content()
            return html_content
    except Exception as err:
        print(f"Error fetching page: {url_parse_result.geturl()}")
        print(err)
        return ""
```

```
# Example usage:
async def main():
    url = "www.example.com/path/to/resource"
    parsed_url = urlparse('https://' + url)

    async with async_playwright() as p:
        browser = await p.chromium.launch()
        html_content = await get_javascript_html(browser, parsed_url)
        print(html_content)
        await browser.close()

await main()
```

Different depths for web crawling were used, and finally, the crawling were executed to **depth 2**, since crawling to that depth returned found url results ranging from 1 urls to around 30 urls and possibly more.

The process took 14 days in total, with 16,510 urls successfully scraped and 14,927 urls corrupted or missing out of 31,427 in the first 7 days. After rescraping after another 7 days, 20,798 urls were successfully scraped as another 15,516 urls were of a no longer functioning sites or redirected urls. The code for this process can be accessed here: MotoLaboInternship2024_webCrawler.ipynb.

Data Summarisation

There are 54,215 companies with juristic IDs, with 52,810 of them being companies without patents, 907 of them being companies with patents and 498 that we couldn’t confirm if they’re patents that matched with the companies or not because the patents were registered by individuals. The individual names could be matched with the committees of companies but we still couldn't check if they are really the committees or not because names could be replicated with someone else that is not really the committees of companies. To this extent we had 907 companies with patents, there are 893 that we could find the names in our databases and 14 that we couldn't find but they could be found at DBD DataWarehouse+.

Topic modelling

To gain deeper insights of the provided patent data, i.e., in how many category can they be classified, topic modelling were performed on the patent using **Latent Dirichlet Allocation (LDA)**, an unsupervised machine learning technique used in clustering and visualising the underlying pattern of a dataset. First of all, libraries and modules required for this task must be installed and imported. The pre-made LDA codes and functions (gensim.models.LdaModel()) used in this process is of **gensim**'s (Řehůřek & Sojka, 2010).

```
!pip install attacut pyLDAvis pythainlp
import gensim
import gensim.corpora as corpora
from gensim.models import CoherenceModel
import matplotlib.pyplot as plt #(Hunter, 2007)
import pandas as pd # (The pandas development team, 2020)
import pythainlp
from pythainlp import word_tokenize
from pythainlp.corpus import thai_stopwords
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
import re
import shutil #https://docs.python.org/3/library/shutil.html
import string #https://docs.python.org/3/library/string.html
import zipfile #https://docs.python.org/3/library/zipfile.html
```

Dataset

Patent data were provided in a total of 164,585 patents, each can be identified using their own unique "app_no" (application number). Precisely, the title and abstract of each patent were used as the main content from which to gain insight. Furthermore, **patent titles and abstracts were used to train LDA models both separately** (labelled as title/abstract) **and together** (labelled as patent)

Data preparation

The raw patent texts in both title and abstract were in natural language. That is, the texts consisted of all kinds of words and symbols. In order to properly extract or interpret any insight from the data without interruptions of noises or corrupted data, the texts must be cleansed of all the noises first. First and foremost step, the patent title and/or abstract that is not of Thai characters were removed, then the numerals and symbols, all of which were filtered out using Regular Expression. Secondly, the resulting data were then tokenised using **PyThaiNLP**'s (Wannaphong Phatthiyaphaibun et al., 2023) word_tokenize() function . Details on the tokeniser used in this process will be discussed in the next part of this process. Finally, stopwords, words that play a crucial role syntactically but not semantically, i.e., have its importance in constructing a sentence while not contributing much in core meaning of the sentence, were removed. The list of Thai stopwords that was used to identify and filter out stopwords in the patent data were downloaded from PyThaiNLP library.

```
# Given a dataframe of thai patents 'patentDF' with 'app_no' 'title' and
'abstract' columns, this part uses Regular Expression to filter out
alphanumeric characters, punctuations, and symbols on the data frame.
thai_patentDF['abstract'] =
thai_patentDF['abstract'].str.replace(r'[a-zA-Z0-9\[\]\(\)\{\}\<\>\_\-\.\\\/
\;!@#$%^&*]+', '', regex=True)
thai_patentDF['title'] =
thai_patentDF['title'].str.replace(r'[a-zA-Z0-9\[\]\(\)\{\}\<\>\_\-\.\\\/\;!
@#$%^&*]+', '', regex=True)
thai_patentDF['abstract'] =
thai_patentDF['abstract'].str.lstrip().str.rstrip()
thai_patentDF['title'] = thai_patentDF['title'].str.lstrip().str.rstrip()
```

```
# Finally, create a new dataframe to contain only rows with either thai
title or
patentTH = thai_patentDF[(thai_patentDF['title'].str.contains(r'[ก-๙]
+')) | (thai_patentDF['abstract'].str.contains(r'[ก-๙]+'))]
thaiPatent
```

```
# Here we define a tokenisation function.
stopwordsTH = thai_stopwords()

def tokenise_thai_text(text, tokeniser):
    # Tokenize the text using pythainlp.word_tokenize
    tokens = pythainlp.word_tokenize(text, engine=tokeniser,
keep_whitespace=False)

    # Remove stop words
    stop_words = thai_stopwords()
    tokens = [token for token in tokens if token not in stop_words]
    # Remove punctuation marks
    tokens = [token for token in tokens if token not in string.punctuation]

    return tokens
```

```
engine = 'attacut' # Put the tokeniser name here; newmm or attacut. Here we
use attacut as an example

TokenisedDF = patentTH.copy()
TokenisedDF.loc[:, 'title'] = TokenisedDF['title'].apply(lambda
text:tokenise_thai_text(text, tokeniser=engine))
TokenisedDF.loc[:, 'abstract'] = TokenisedDF['abstract'].apply(lambda
text:tokenise_thai_text(text, tokeniser=engine))
TokenisedDF['title_abstract'] = TokenisedDF['title'] +
TokenisedDF['abstract']
TokenisedDF['title_abstract'] = TokenisedDF['title_abstract'].apply(lambda
x: [word for word in x if word != ''])
```

Feature Engineering

For the training of LDA models, two alternatives for feature extraction were Bag-of-words representation and TF-IDF vector representation.

Hyperparameter Tuning

Like any other machine learning model, to get to the optimal result, hyperparameter tuning is inevitable. Here we discuss how we compared topic modelling results from LDA models of different combinations of parameters.

The parameters tried in tuning were of the following:

1. Tokeniser: also known as “engine”, two tokenisers were chosen; (1) **NEWMM**, a tokeniser which implement maximal matching algorithm on Thai text tokenisation and (2) **AttaCut**, a fast and reliable deepcut tokeniser that compensates accuracy with speed, making it less accurate as compared to other deepcut tokeniser. Nonetheless, it is objectively more accurate than non-deepcut tokeniser, in this case, the NEWMM. Both tokenisers are applicable with PyThaiNLP’s word_tokenize() function
2. Number of topics: To train an LDA model, the number of topics into which the data will be classified must be set. As there is no exact answer for how many topics should be set, hence the

tuning is required. Here, different numbers of topics were compared against each other with coherence score calculated using **CoherenceModel** from **gensim.models** as performance measure between numbers of topics ranging from 2 to 40 topics.

Note that number of passes can also be tuned, but, due to hardware and time limitations, the number of passes were set to 1 during number of topics tuning, then the final models were then trained with their best number of topic with 10 passes.

As the bag-of-words represented models were trained first on title and abstract data separately, the decision of whether the best tokeniser for topic modelling is NEWMM or AttaCut was made before the training of TF-IDF represented models and before the training on concatenated title and abstract data, as patent data were trained separately first. Finally, the best tokeniser for topic modelling was AttaCut as the tokenisation made with the engine were more accurate and usable, while the results from models trained using NEWMM consisted many butchered technical words as affected by NEWMM's inefficiency with technical words. The TF-IDF models were then trained with solely AttaCut tokenised data, while the bag-of-words NEWMM tokenised patent data were not trained.

<u>Training and results</u>

LDA models were trained with different combinations of hyperparameters as stated above, as well as feature extraction alternatives. Let us cover some important steps in training an LDA model. Here is an example of bag-of-words represented patent data were trained:

Let's first create a word-to-id dictionary and a bag-of-words corpus.

```
patent_content = 'title_abstract' # Replace with your input data; title, abstract, or title_abstract. Here, for example, we use 'title_abstract'.

input_data = TokenisedDF[patent_content]

# Create a Gensim Dictionary from tokenized documents
ldadict = gensim.corpora.Dictionary(input_data)

# Convert the term-document matrix to a list of BoW (bag-of-words) format
ldacorpus = [ldadict.doc2bow(doc) for doc in input_data]
```

In case of TF-IDF representation, after the dictionary and corpus has been created, use the BOW corpus to create a TF-IDF model and ultimately, TF-IDF corpus, then replace any "corpus" parameter in the code that follows.

```
# Create a TF-IDF model from the corpus
tfidf = gensim.models.TfidfModel(ldacorpus)

# Convert the corpus to TF-IDF format
tfidf_corpus = tfidf[ldacorpus]
```

With the dictionary and the corpus created, let's first find the best number of topics for the model. To save time, the number of passes is set to 1.

```
# Function to compute coherence score
def find_best_numTopic(dictionary, corpus, texts, limit=10, start=1,
step=1):
    coherence_values_dict = dict()
    for num_topics in range(start, limit+1, step):
        print('initiating training at',num_topics,'topics')
        model = gensim.models.LdaModel(corpus=corpus, id2word=dictionary,
passes=1, num_topics=num_topics, random_state=42)
        coherencemodel = CoherenceModel(model=model, texts=texts,
dictionary=dictionary, coherence='c_v')
        coherence_values_dict[num_topics] = coherencemodel.get_coherence()
        print('finished training at',num_topics,'topics. Coherence
score:',coherence_values_dict[num_topics])
    return coherence_values_dict

limit = 50  # max number of topics
start = 2   # min number of topics
step = 1    # step size
numTopic2coherence = find_best_numTopic(dictionary=ldadict,
                                              corpus=ldacorpus,
                                              texts=input_data,
                                              start=start,
                                              limit=limit,
                                              step=step)
```

```
# The results can be visualised on a graph.

# Create line plot
plt.figure(figsize=(10, 6))
plt.plot([str(int(xi)) for xi in numTopic2coherence.keys()],
numTopic2coherence.values(), marker='o', linestyle='-')

# Add labels and title
```

```
plt.xlabel('Number of Topics', fontsize=12)
plt.ylabel('Coherence Value', fontsize=12)
plt.title(f'Coherence Plot: {patent_content} (2-50)', fontsize=14)

# Show the plot
plt.grid(axis='y', alpha=0.5)
plt.show()
```

Sort the dictionary items by value in descending order and select the top , say, three.

```
top_num = 3 # Put the number of top n here as you want.

topN_cohe = sorted(numTopic2coherence.items(), key=lambda item: item[1],
reverse=True)[:top_num]
bow_topN = dict()

# Print the top three key-value pairs
for key, value in top3_cohe:
  topN[key] = value
  print(f'{key}: {value}')
```

Next, we train three LDA models with the most coherence scores on 10 passes to finally come to the most optimal result of the combination.

```
# Function to compute coherence score
def runNcompute_spec_topNum(dictionary, corpus, texts, spec_topNum,
passes_num):
    coherence_values_dict = dict()
    model_list = []
    if isinstance(spec_topNum, list) or isinstance(spec_topNum, set):
      for num_topics in spec_topNum:
          print('initiating training at',num_topics,'topics')
          model = gensim.models.LdaModel(corpus=corpus, id2word=dictionary,
passes=passes_num, num_topics=num_topics, random_state=42)
          model_list.append(model)
          coherencemodel = CoherenceModel(model=model, texts=texts,
dictionary=dictionary, coherence='c_v')
          coherence_values_dict[num_topics] = coherencemodel.get_coherence()
          print('finished training at',num_topics,'topics. Coherence
score:',coherence_values_dict[num_topics])
      return model_list, coherence_values_dict
    elif isinstance(spec_topNum, int):
      print('initiating training at',spec_topNum,'topics')
      model = gensim.models.LdaModel(corpus=corpus, id2word=dictionary,
passes=passes_num, num_topics=spec_topNum, random_state=42)
```

```
      model_list.append(model)
      coherencemodel = CoherenceModel(model=model, texts=texts,
dictionary=dictionary, coherence='c_v')
      coherence_values_dict[spec_topNum] = coherencemodel.get_coherence()
      print('finished training at',spec_topNum,'topics. Coherence
score:',coherence_values_dict[spec_topNum])
      return model_list, coherence_values_dict
    else:
      raise ValueError("spec_topNum must be a list, a set, or an integer")

CandidModels10p, bowCandidCoherence10p =
runNcompute_spec_topNum(dictionary=ldadict,

corpus=ldacorpus,

texts=input_data,

spec_topNum=sorted(topN.keys()),

passes_num=10)
```

```
# Create line plot
plt.figure(figsize=(10, 6))
plt.plot([str(int(xi)) for xi in CandidCoherence10p.keys()],
CandidCoherence10p.values(), marker='o', linestyle='-')

# Add labels and title
plt.xlabel('Number of Topics', fontsize=12)
plt.ylabel('Coherence Value', fontsize=12)
plt.title(f'Coherence Plot: {patent_content} (Candidate)', fontsize=14)

# Show the plot
plt.grid(axis='y', alpha=0.5)
plt.show()
```

To get and save the best model trained earlier, the functions for the task are defined here.

```
def get_best_model(model_list, coherence_values_dict):
  print('finding best model.....')
  max_coherence = max(coherence_values_dict.values())
  for ind, numTopic in enumerate(coherence_values_dict):
    if coherence_values_dict[numTopic] == max_coherence:
      best_model = model_list[ind]
  print('best model found!')
  return best_model

def saveNzip_model(LDAmodel, model_folder):
  print('saving model....')
  # Save the LDA model
  LDAmodel.save(model_folder)


  # List all the files related to the saved LDA model
  model_files = [model_folder + '.model', model_folder + '.state',
model_folder + '.id2word']


  # Create a zip file containing the saved model files
  zip_filename = model_folder + '.zip'
  with zipfile.ZipFile(zip_filename, 'w') as zipf:
      for file in model_files:
          zipf.write(file)
          # os.remove(file)  # Optional: Remove the original file after
zipping


  # Verify that the files are zipped
  print(f"Zipped model files into {zip_filename}")


best_model = get_best_model(CandidModels10p, CandidCoherence10p)
saveNzip_model(best_model, f'{patent_content}LDA')
```

After the best model is retrieved and saved, we visualise the result.

```
# Prepare the visualisation
bow_LDAvis = gensimvis.prepare(bow_best_model, ldacorpus, ldadict)

# Display the visualisation
pyLDAvis.display(bow_LDAvis)
```

Note these snippets represent separate code cells. Therefore, each cell should be run separately and sequentially, to avoid any visual output to be overwritten by some other functions. For instance, the final cell that downloads the LDA visualisation result will prevent the visualisation from the cell earlier to be shown.

```
# save the visualisation
```

```
pyLDAvis.save_html(bow_LDAvis,
f'bow_{patent_content}_lda_visualization.html')
```

Finally, Here are the training results from all the combinations. The code for LDA model training can be found here: PatentLDAtemplate.ipynb.

Feature Representation	Tokeniser	Content	Best number of topics
BOW	NEWMM	title	8
		abstract	8
	AttaCut	title	41
		abstract	13
		patent (title + abstract)	14
TF-IDF	AttaCut	title	11
		abstract	7
		patent (title + abstract)	3

Dual attention

To obtain product information from the scraped website content, we used a dual attention model to identify the relevant webpage containing product details and extract product-related keywords from that page.

Data preparation

We initially had a total of 907 websites with patents from both this year and last year. After preprocessing by removing duplicates, non-Thai patents, patents without content, and patents with mismatched urls, we were left with 634 websites with patents.

For firms without patents, we sampled 800 websites out of 52,810 firms from both this year and last year. We then preprocessed this data by removing duplicates, non-Thai patents, and patents without content, which left us with 574 websites with patents.

We tokenised the content using two different tokenisers: NEWMM and Attacut, then removed stop words from the website content using the PyThaiNLP library.

```
!pip install pythainlp attacut
```

```
import pandas as pd
from pythainlp import word_tokenize
from pythainlp.corpus import thai_stopwords
from pythainlp.tokenize import word_tokenize as thai_word_tokenize

# Example DataFrame as a web content
data = {
    'content': [
        'นี่คือเนื้อหาตัวอย่าง ทดสอบการใช้ PyThaiNLP เพื่อลบ stop words และทำ
tokenization ก่อนเทรนโมเดล',
    ]
}
df = pd.DataFrame(data)

# Function to remove stop words and tokenize content
def process_content(content):

    # Tokenization with two different tokenizers
    tokens_newmm = word_tokenize(content, engine='newmm')
    tokens_attacut = thai_word_tokenize(content, engine='attacut')

    # Remove stopwords
    stopwords = thai_stopwords()
    filtered_words_newmm = [word for word in tokens_newmm if word not in
stopwords]
    filtered_words_attacut = [word for word in tokens_attacut if word
not in stopwords]


    return {
        'filtered_content_newmm': ' '.join(filtered_words_newmm),
        'filtered_content_attacut': ' '.join(filtered_words_attacut),
        'tokens_newmm': tokens_newmm,
        'tokens_attacut': tokens_attacut
    }

# Apply processing function to each row in the DataFrame
df_processed = df['content'].apply(process_content).apply(pd.Series)
```

```
# Display the processed DataFrame
df_processed
```

Next, we embedded all the tokens using two different embeddings: the Thai National Corpus (TNC) with 300 dimensions, which includes 61,630 vocabulary items, and FastText with 300 dimensions, limiting the vocabulary to 800,000 items due to limited memory of Google Colab.

```
from gensim.models import KeyedVectors

th_dictionary = KeyedVectors.load_word2vec_format('./cc.th.300.vec',
limit=800000) # FastText embedding

words = list(th_dictionary.key_to_index)  # Get all word keys
vectors = th_dictionary.vectors  # Get the corresponding vector matrix

# Create the word-to-vector dictionary
wv_dict = dict(zip(words, vectors))
```

<u>Training</u>

We first tested nine different learning rates over 10 and 20 epochs, using four combinations of the two tokenisers and two embeddings mentioned in the data preparation with only 2023 patent data. After we finished processing 2024 patent data, we used 2023 and 2024 patent data together to improve the model performance. The code for training dual attention model can be accessible here NLP intern_dual_attn_revised.ipynb

<u>Results</u>

Here are the parameters that achieved the highest accuracy for each combination.

Tokeniser	**Embedding**	**Learning rate**	**Eppalyoch**	**Patent data**	**Accuracy**
NEWMM	Thai National Corpus	0.05	10	2023 only	0.675
NEWMM	FastText	0.5	10	2023 only	0.693
Attacut	Thai National Corpus	0.2	10	2023 only	0.668
Attacut	FastText	0.5	10	2023 only	0.682
NEWMM	FastText	0.7	20	2023 and 2024	0.7

The best model achieved an accuracy of 0.7 using the NEWMM tokenizer and FastText embedding, with a learning rate of 0.7 and 20 epochs. The results of dual attention model can be accessible here dual_attention_result.csv

Concordance matrix

To find the potential of technology from websites with patent information, we used the concordance matrix (Patent2Product matrix) to transform technology information into market information.

Data preparation

For market vectors, we used the product-related keywords from the results of the dual attention model, resulting in 612 rows of keywords from 634. The missing 22 rows contained only 1-2 repetitive words in the content. There are 539 companies with page weights greater than or equal to 0.1; for these companies, we gathered 1,570 keyword lists and combined each list of keywords corresponding to each company. For the 73 companies with page weights less than 0.1, we used only one list of keywords with the highest weight. Finally, we embedded all the keywords into market vectors by averaging the word embeddings of each company's keywords.

For technology vectors, we first matched the company names with a patent database using the Department of Intellectual Property website to obtain the patent application numbers. We retrieved 2,028 patents from 565 companies out of 612, with 47 companies not matching. We then used the application numbers to find the titles and abstracts of the patents and concatenate them into one column. We preprocessed this concatenated column by tokenizing with AttaCut, removing stop words with PyThaiNLP, and removing non-Thai rows with regular expressions (RegEx) .

```
!pip install pythainlp attacut
```

```python
import pandas as pd
import regex as re
from pythainlp.tokenize import word_tokenize
from pythainlp.corpus import thai_stopwords

# Example DataFrame with 'title_abstract' column
data = {
    'title_abstract': [
        'This is an English title and abstract.',
        'นี่คือเนื้อหาและบทคัดย่อภาษาไทย ทดสอบการใช้ PyThaiNLP.',
        'Another ภาษาไทย example with mixed languages.'
    ]
}
df = pd.DataFrame(data)

# Thai stopwords
stopwords = thai_stopwords()

# Function to preprocess text
def preprocess_text(text):
    # Use regex to remove non-Thai rows
    thai_pattern = re.compile(r'[\u0E00-\u0E7F]+')  # Thai Unicode range
    text_thai = ' '.join(re.findall(thai_pattern, text))

    # Tokenize with AttaCut tokenizer
    tokens = word_tokenize(text_thai, engine='attacut')

    # Remove stopwords
    filtered_tokens = [word for word in tokens if word not in stopwords]

    return ' '.join(filtered_tokens)

# Apply preprocessing function to 'title_abstract' column
df['processed_title_abstract'] =
df['title_abstract'].apply(preprocess_text)

# Display the processed DataFrame
df
```

Finally, we embedded all the tokens into technology vectors by averaging the word embeddings for each patent. If a company had more than one patent, we aggregated the vectors by averaging them.

Training

With a total of 612 rows for the market vectors and 565 rows for the technology vectors, we needed to equalise the rows to 565 to train the concordance matrix. We then split the data into a training set and a test set in a 9:1 proportion, resulting in 508 rows for the training set and 57 rows for the test set. We tested the model using seven different learning rates over 40 epochs. The code is provided Building_concordance_matrices_revised.ipynb

Results

The best model achieved an average cosine similarity of 0.51 on the test set, with a learning rate of 0.7 and 40 epochs.

Product suggestion

After we have trained Patent2Product model. We would like to implement it on real-world data which in this situation is Chulalongkorn University's patents. Because Chulalongkorn University does not do business using its patents, we could use this model to find the market opportunity based on the patents.

Data preparation

We were provided with Chulalongkorn university's patent in a total of 660 patents. We first preprocessed the provided patent data by tokenizing with Attacut, removing stop words with PyThaiNLP, and removing non-Thai rows with regular expressions (RegEx) using the same code provided in the technology vectors section. We were left with 621 patents.

Topic modelling

To gain deeper insights into the provided patent data, we performed topic modelling using Latent Dirichlet Allocation (LDA). We tested the number of topics from 2 to 20, and the optimal number of topics with the highest coherence score was 6. The six topics identified are: 1) Structural Components, 2) Chemical Reactions, 3) Medical and Diagnostic Procedures, 4) Material Composition and Mixtures, 5) Extraction and Processing, and 6) Biological and Agricultural Products. The code for Topic modelling can be accessed at LDA chula patent.ipynb

Results

Based on topic modelling, we had six topics and we decided to predict the product by a patent for each topic. We've selected 6 patents and predicted its estimated market vectors. The estimated vectors couldn't directly be interpreted but they were vectors that in word embedding form (300 dimensions). Therefore, we could find its cosine similarity of vocabularies in word embedding or compare it with firms' market vectors which are averages of word embeddings. We selected top 20 similar words from word embeddings' vocabularies and top 3 similar market vectors from firms. The code is provided Concordance_matrix_predict_chula_patent.ipynb

This table represents the results of performing a concordance matrix on Chula's patent. The results is displayed in 6 columns 1) Topic: the 6 topics from the results of topic modelling 2) Application No: application number of the patent 3) Title: title of the patent 4) Similar product keyword with cosine similarity: top 20 similar product keywords. 5) Similar firm with cosine similarity: top 3 similar firms' market vector 6) Firm keywords: keywords from top similar firms.

The full results of all patents can be accessible at concordance_matrix_results

Topic	**Application No.**	**Title**	**Similar product keyword with cosine similarity**	**Similar firm with cosine similarity**	**Firm keywords**
Structural Components	801002377	Dust collector	บริการ (service), บริหาร จัดการ(management), ผลิต (produce), ขนส่ง (transport), ติดตั้ง (install), อุตสาหกรรม (industry), แจกจ่าย (distribute), สะดวก (convenient), จัดหา (procure), เทคโนโลยี สารสนเทศ (information technology), บริโภค (consume), อุปโภค บริโภค (consumption), ดูดซับ (absorb), อยู่ อาศัย (reside), สินเชื่อ (credit), ข้อมูล (data), สาธารณูปโภค (utilities), อุปกรณ์	1. บริษัท ดี ซูพรีม จำกัด (0.7977129836479814) 2. บริษัท บิวเดอสมาร์ท จำกัด (มหาชน) (0.7930728213578875) 3. บริษัท ไซท์ เพรพพาเรชั่น แมเนจเมนท์ จำกัด (0.7732154808001884)	1. เชี่ยวชาญ (expert), บริการ (service), ข่าวสาร (news), จีน (Chinese), สากล (international), รอบคอบ (comprehensive), สมาชิก (member), สมบูรณ์ (complete), ให้การ (provide), มลพิษ (pollution), เหมาะ (suitable), เสนอ (offer), สะอาด (clean), ลม (wind), ... 2. คำนวณ (calculate), เรียบร้อย (neat), กรอก (fill), ทำการ

			(equipment), ตรวจสอบ (inspect), ชลประทาน (irrigation)		(conduct), ดูแล (take care), แห้ง (dry), เลือก (choose), ต้อนรับ (welcome), เซนติเมตร (centimeter), รอย (trace), ปราด (remove), บริการ (service), ปรึกษา (consult), ระบบ (system), ตอบ (answer), ... 3. แจ้ง (notify), ตู้ (cabinet), สถาน (place), แทบ (almost), เชื่อม (connect), บริเวณ (area), กระแทก (impact), ดูแล (take care), กรอก (fill), ทำการ (conduct), ซึม (absorb), ลม (air), เลือก (choose), การนำ (leading), พนักงาน (staff), ...
Chemical Reactions	1703000392	Process for increasing the flexibility of hydrogel sheets prepared by freeze-thawing with proteinaceous adhesive sericin	ชุ่มชื้น (moist), เจือจาง (dilute), ปริมาณ (quantity), เกลือแร่ (minerals), บริการ (service), ปนเปื้อน (contaminated), ออกซิเจน (oxygen), แอลกอฮอล์ (alcohol), ชุ่มชื่น (moist), สมุนไพร (herbs), จุลินทรีย์ (microorganisms), ดูดซับ (absorb), พลังงาน (energy), ย่อยสลาย (decompose), กลูโคส (glucose), อินซูลิน (insulin), เมือก (mucus), แคลเซียม (calcium), เส้นใย (fiber)	1. บริษัท ทีพีไอ โพลีน จำกัด (มหาชน) (0.8180409333119143) 2. บริษัท เอเชีย แปซิฟิก ดริลลิ่ง เอ็นจิเนียริ่ง จำกัด (0.7868153343935619) 3. บริษัท ทีโอเอ เพ้นท์ (ประเทศไทย) จำกัด (มหาชน) (0.7809962732083628)	1. แอลกอฮอล์ (alcohol), ประตู (door), ปริมาณ (quantity), บำบัด (treatment), วิเคราะห์ (analysis), กระบวนการ (process), ประชุม (meeting), วิจัย (research), ทดสอบ (test), ที่มา (origin), กลไก (mechanism), จำหน่าย (distribution), ทำร้าย (damage), ปล่อย (release), วิตามิน (vitamin), ... 2. เลือก (choose), โขนง (pest), ป้องกัน (prevent), คุ้มครอง (protect), กำจัด (eliminate), ผัก

					(vegetable), ประสิทธิภาพ (efficiency), เคมี (chemistry), ประเภท (type), สมดุล (balance), เคมีภัณฑ์ (chemical products), ยา (medicine), วิธี (method), สาเหตุ (cause), ปัญหา (problem), ... 3. เคมี (chemistry), ประเภท (type), ตัวอย่าง (example), วิดีโอ (video), ปัญหา (problem), เต็ม (full), วิเคราะห์ (analyze), บทความ (article), บริเวณ (area), ดัน (force), บวม (swell), ทิ้ง (discard), เจอ (encounter), รอย (trace), พิจารณา (consider), ...
Medical and Diagnostic Procedures	1601006353	The set of oligonucleotides and synthetic DNA analogues for genetic testing and genetic testing kits for detecting bacteria, listeria, and monocytogenes in food	ปฐมพยาบาล (first aid), ร่างกาย (body), ระคายเคือง (irritation), รักษาพยาบาล (medical treatment), บริการ (service), ขับถ่าย (excretion), ผ่าตัด (surgery), ภูมิคุ้มกัน (immunity), เจ็บป่วย (illness), บำบัด (therapy), ป่วย (sick), ชุ่มชื้น (moisture), ออกซิเจน (oxygen), ดูแลรักษา (care), สุขภาพ (health), โรค (disease), อินซูลิน (insulin), ปฏิชีวนะ (antibiotic), ป้องกัน (prevention), อักเสบ (inflammation)	1. บริษัท เบรน ไดนามิกส์ เทคโนโลยี จำกัด (0.8139842510282688) 2. บริษัท เฮเว่น เฮิร์บ จำกัด (0.8051043131495222) 3. บริษัท ปูนซิเมนต์ไทย (แก่งคอย) จำกัด (0.7988034566672856)	1. ผ่าตัด (surgery), วิจัย (research), ร่างกาย (body), รักษา (treat), ปลอดภัย (safe), ตรวจ (check), สุขภาพ (health), โรค (disease), แพทย์ (doctor), หลับ (sleep), วินิจฉัย (diagnose), ภาวะ (condition), หาย (recover), บำบัด (therapy), นัด (appointment), ... 2. ป้องกัน (prevention), สุขภาพ (health), บำบัด (therapy), แม่นยำ (accurate), แพ้ (allergy), ยา (medicine), ตรวจ (check), แผล (wound), ผื่น (rash), ไข้ (fever),

					อาการ (symptom), โรค (disease), แพทย์ (doctor), เวชภัณฑ์ (medical supplies), ร่างกาย (body), ... 3. กายภาพ (physical therapy), มะเร็ง (cancer), ป้องกัน (prevention), ปรึกษา (consult), รักษา (treat), ปลอดภัย (safe), สุขภาพ (health), เสี่ยง (risk), แก้ (solve), สาเหตุ (cause), ผื่น (rash), อาการ (symptom), โรค (disease), แพ้ (allergy), ภูมิแพ้ (allergic), ...
Material Composition and Mixtures	1503000655	Geopolymer from fly ash and aluminium waste	ชุ่มชื้น (moisture), บริการ (service), ปริมาณ (quantity), ปนเปื้อน (contamination), ดูดซับ (absorb), เกลือแร่ (mineral salts), ย่อยสลาย (decompose), ข้อมูล (data), ขวัญกำลังใจ (morale), ผลิต (produce), บริโภค (consume), ออกซิเจน (oxygen), ขนส่ง (transport), พลังงาน (energy), ปฐมพยาบาล (first aid), จุลินทรีย์ (microorganisms), คุณภาพ (quality), เจือจาง (dilute), คัดแยก (separate), ปฏิชีวนะ (antibiotic)	1. บริษัท ไซท์ เพรพพาเรชั่น แมเนจเมนท์ จำกัด (0.8568271132498728) 2. บริษัท ทีโอเอ เพ้นท์ (ประเทศไทย) จำกัด (มหาชน) (0.8273646821182727) 3. บริษัท บิวเดอสมาร์ท จำกัด (มหาชน) (0.7975129664971969)	1. อิสระ (freedom), ดูแล (take care), แห้ง (dry), เวลา (time), คุ้มครอง (protect), เงินปันผล (dividend), ข่าวสาร (news), ประเมิน (evaluate), บทบาท (role), ลักษณะ (characteristic), ภาวะ (condition), จ่าย (pay), คัด (select), เบา (light), วัตถุดิบ (ingredients), ... 2. ประปา (water supply), มลภาวะ (pollution), แจ้ง (notify), ปลวก (termite), เชื่อม (connect), หัวข้อ (topic), คำนวณ (calculate), พัด (fan), เสนอ (propose), ดูแล (take care), แผนผัง (diagram), ซึม (absorb), แล็บ (laboratory), เลือก (choose), ตึง (tense), ...

					3. คำนวณ (calculate), เรียบร้อย (neat), กรอก (fill), ทำการ (conduct), ดูแล (take care), แห้ง (dry), เลือก (choose), ต้อนรับ (welcome), เซนติเมตร (centimeter), รอย (trace), ปราด (remove), บริการ (service), ปรึกษา (consult), ระบบ (system), ตอบ (answer), ...
Extraction and Processing	1403000772	Sugar-free protein drink formulated with rice bran extract	ชุ่มชื้น (moisture), เกลือแร่ (mineral salts), น้ำยา (solution), แอลกอฮอล์ (alcohol), เสมหะ (phlegm), เมือก (mucus), ยาบำรุง (tonic), โยเกิร์ต (yogurt), สมุนไพร (herbs), หมัก (ferment), โลชั่น (lotion), อาหาร (food), ยา (medicine), ในเวลา (in time), จุลินทรีย์ (microorganisms), ซีอิ๊ว (soy sauce)	1. บริษัท เฮเว่น เฮิร์บ จำกัด (0.8370235267455878) 2. บริษัท อินเตอร์เนชั่นแนล คอมพิวติ้ง แอนด์ เน็ตเวอร์กกิ้ง จำกัด (0.824167733320119) 3. บริษัท กิฟฟารีน สกายไลน์ ยูนิตี้ จำกัด (0.8222320829810246)	1. สมุนไพร (herbal), ศักยภาพ (potential), จ้าง (hire), สะอาด (clean), ป้องกัน (prevent), ทำการ (conduct), ประกัน (insurance), แห้ง (dry), กำจัด (eliminate), เย็น (cool), โอกาส (opportunity), บริการ (service), สะดวก (convenient), สุขภาพ (health), อาหาร (food), ... 2. กุ้ง (shrimp), ปลอด (safe), สุขภาพ (health), ชู (boost), โรค (disease), สุกร (pigs), เลือก (choose), โภชนาการ (nutrition), ค้นหา (search), เร่ง (accelerate), ทาน (eat), สุข (happy), ห้อง (room), เดิน (walk), ... 3. ค้นหา (search), เค็ม (salty), น้ำตาล (sugar), สุขภาพ (health), เต็ม (full), ใส่ (put), แป้ง (flour), แก้ (cure), รู (hole), ช็อก

					(shock), ยา (medicine), ซา (sour), หวาน (sweet), เย็น (cool), ร่างกาย (body), ...
Biological and Agricultural Products	903000683	Probiotic-enriched brown rice pudding, dairy-free and sugar-free	ซีอิ๊ว (soy sauce), โยเกิร์ต (yogurt), มะละกอ (papaya), หมัก (ferment), เต้าหู้ (tofu), สมุนไพร (herbs), ลูกตาล (palmyra fruit), หน่อไม้ (bamboo shoots), ชุ่มชื้น (moisture), กะปิ (shrimp paste), กลูโคส (glucose), น้ำยา (solution), น้ำพริกเผา (chili paste), เต้าเจี้ยว (fermented soybean paste), ชาเขียว (green tea), ยาบำรุง (tonic), ถั่วลิสง (peanuts), ปลากะพง (seabass), มะเขือเทศ (tomato), ซ้อสปรุงรส (seasoning sauce)	1. บริษัท อินเตอร์เนชั่นแนล คอมพิวติ้ง แอนด์ เน็ตเวอร์กกิ้ง จำกัด (0.8169769806841805) 2. บริษัท กิฟฟารีน สกายไลน์ ยูนิตี้ จำกัด (0.8056225980016989) 3. บริษัท อายิโนะโมะโต๊ะ เบทาโกร โฟรเซ่นฟู้ดส์ (ประเทศไทย) จำกัด (0.7914237028305982)	1. กุ้ง (shrimp), ปลอด (safe), สุขภาพ (health), ชู (boost), โรค (disease), สุกร (pigs), เลือก (choose), โภชนาการ (nutrition), ค้นหา (search), เร่ง (accelerate), ทาน (consume), สุข (well-being), ห้อง (room), เดิน (walk), อิ่ม (satisfied), ... 2. ค้นหา (search), เค็ม (salty), น้ำตาล (sugar), สุขภาพ (health), เต็ม (full), ใส่ (put), แป้ง (flour), แก้ (cure), รู (hole), ช็อก (shock), ยา (medicine), ซา (sour), หวาน (sweet), เย็น (cool), ร่างกาย (body), ... 3. ร่างกาย (body), บริโภค (consume), แนวทาง (guideline), อาทิ (such as), ม้วน (roll), โภชนาการ (nutrition), เลือก (choose), นึ่ง (steam), ต้มยำ (spicy soup), เกร็ด (grain), ต้ม (boil), สุข (well-being), ผัก (vegetables), น้ำตก (waterfall), บริการ (service), ...

Appendix

Web Scraping Data dict

The file can be accessed at scraped_2023_2024_path.csv

Field Name	Data Type	Format	Description	Constraints	Example
juristic_id	integer	12 digits	a 12-digit identification number searchable on DBD warehouse database	unique, not null	105532025192
path	string	filepath	path of the scraped html content	must be an absolute path	2024_Project/Internship_files/TH_Suchit/Organized/4000_4500/ 2conlinestore.com.tar.gz
name	string	N/A	name of the firm		บริษัท ทูซีบรอดคาส จำกัด
domain	string	url domain	domain of the scraped website	not null	2conlinestore.com
is_patent	integer	1 digit	a status represented if the company own any patent as a single-digit integer. 0 mean not patent, 1 mean patent and 2 mean not sure because there are some individuals who are companies ‘ committee whose names could be matched with patent applicants	not null	0
year	integer	4 digits	year that the file was scraped	not null	2024
biz_size	string	VARCHAR(1)	business size of the firm represented as a capitalised alphanumeric character. There are 4 types S, M, L , X corresponding to small, medium, large, and extra respectively.		S

biz_type	string	VARCHAR(1)	business type of the firm represented as a capitalised alphanumeric character. The criteria of categorising could be found in Thailand Standard Industrial Classification : TSIC-2009 or การจัดประเภทธุรกิจของนิติบุคคล โดย กรมพัฒนาธุรกิจการค้า		G
biz_size_type	string	VARCHAR(2)	concatenated of business size and business type of the firm		SG

Concordance matrix results Data dict

The file can be accessed at concordance_matrix_results

Field Name	**Data Type**	**Format**	**Description**	**Constraints**	**Example**
app_no	integer		applicant identification number searchable on Department of Intellectual Property IP search	unique, not null	1003166
ayear	integer	4 digits	year of patent filing	not null	2000
raw_title_abstract	string	N/A	raw concatenated text of a patent's title and abstract	not null	เครื่องมือสำหรับเตรียมชิ้นงานที่ผลิตด้วยโพลิยูรีเทนในถังปฏิกรณ์แบบกะขนาดเล็ก DC60 เครื่องมือตามการประดิษฐ์นี้ ใช้สำหรับเตรียมชิ้นงานที่ทำจากโพลิยูรีเทนโฟม (Polyurethane Foam)...
top20	list	N/A	a list of tuples of top 20 of	not null, list of	[('บริการ',

			similar product keywords with cosine similarity	tuples of string and float	0.594870924949646), ('ข้อมูล', 0.46171653270721436), ('รักษาพยาบาล', 0.4088881313800812), ...)]
cos_sim	list	N/A	a list of tuples of top3 most similar firm's juristic_id and each firm's cosine similarity	not null, list of integer and float	[(107550000173, 0.783358237390702), (107560000133, 0.7797444769957901), (125550039833, 0.7786302535701481)]
cos_sim_firm_1	list	N/A	a list of product keywords of the firm with the highest cosine similarity extracted using dual-attention model	not null, list of strings	['เพิง', 'ถูกใจ', 'น็อค', 'คำนวณ', 'เรียบร้อย', 'กรอก', 'ทำการ', 'ดูแล', 'แห้ง', 'มะนิลา', 'เลือก', 'ต้อนรับ', 'เซนติเมตร', 'รอย', 'ปราด', 'บริการ', ...]
cos_sim_firm_2	list	N/A	a list of product keywords of the firm with second highest cosine similarity extracted using dual-attention model	not null, list of strings	['ประปา', 'มลภาวะ', 'แจ้ง', 'ปลวก', 'เชื่อม', 'หัวข้อ', 'คำนวณ', 'พัด', 'เสนอ', 'ดูแล', 'แผนผัง', 'ซึม', 'แล็บ', 'เลือก', 'ตึง', 'คู่', 'ระบุ', 'กำจัด', 'บริการ', ...]
cos_sim_firm_3	list	N/A	a list of product keywords of the firm with third highest cosine similarity extracted using dual-attention model	not null, list of strings	['เหมาะ', 'หน้า', 'ป้องกัน', 'กระแทก', 'จำหน่าย', 'เลท', 'คู่', 'เข่ง', 'ทิ้ง', 'เย็น', 'ผัก', 'ประโยชน์', 'ข่าวสาร', 'คลิก', 'แนบ', 'ต้องการ', 'ลักษณะ', 'ภาวะ', 'แม่นยำ', 'ปลอดภัย',

					...]

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

Microsoft. (2020). Fast and reliable end-to-end testing for modern web apps | Playwright. Playwright.dev. Retrieved from https://playwright.dev/

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

Python Software Foundation. (n.d.). shutil — High-level file operations — Python 3.12.4 documentation. Retrieved from https://docs.python.org/3/library/shutil.html

Python Software Foundation. (n.d.). string — Common string operations — Python 3.12.4 documentation. Retrieved from https://docs.python.org/3/library/string.html

Python Software Foundation. (n.d.). urllib — URL handling modules — Python 3.12.4 documentation. Python.org. Retrieved from https://docs.python.org/3/library/urllib.html

Python Software Foundation. (n.d.). zipfile — Work with ZIP archives — Python 3.11.4 documentation. Retrieved from https://docs.python.org/3/library/zipfile.html

Regular Expressions. (n.d.). Retrieved from https://www.regular-expressions.info/

Řehůřek, R., & Sojka, P. (2010, May 22). Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 45–50. Valletta, Malta: ELRA.

Richardson, L. (2007). Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation. Crummy.com. Retrieved from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

The pandas development team. (2020, February). pandas-dev/pandas: Pandas (latest) [Software]. Zenodo. https://doi.org/10.5281/zenodo.3509134

Wannaphong Phatthiyaphaibun, Korakot Chaovavanich, Charin Polpanumas, Arthit Suriyawongkul, Lalita Lowphansirikul, Pattarawat Chormai, Peerat Limkonchotiwat, Thanathip Suntorntip, and Can Udomcharoenchaikit. 2023. PyThaiNLP: Thai Natural Language Processing in Python. In Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023), pages 25–36, Singapore. Association for Computational Linguistics.