Heimdallr

The guardian API for academic integrity

By Tomas Sanchez

For Universidad Tecnológica Nacional, Facultad Regional Buenos aires

Natural Language Processing, K3551

As taught by Eng. Hernan Borre

November 10th, 2023

Table of Contents

Abstract

In this research, a production-ready REST API was developed to present plagiarism detection as a service. For this, the latest, most modern, asynchronous compatible Python framework was used in conjunction with pre-trained natural language processing (NLP) models for Spanish texts. Writings are compared by sentence against previously stored sentences from different works. Plagiarism is determined by cosine similarity, with a configurable threshold.

Keywords: NLP, REST API, Python, Cosine Similarity, Asynchronous Processing, Command query responsibility segregation (CQRS), Event driven, Plagiarism Detection.

Introduction

Heimdallr is A powerful and intelligent plagiarism detection system designed to uphold academic integrity and ensure the authenticity of written works.

Inspired by the mythological figure Heimdallr, the all-seeing, known for his unwavering vigilance. Its advanced detection capabilities and user-friendly interface result in the ultimate solution for educational institutions, researchers, and students to maintain the highest standards of academic honesty and originality.

This plagiarism detector application was developed as an assignment for Natural Language Processing course from Universidad Tecnologica Nacional, Facultad Regional Buenos Aires during the second semester of 2023.

Features

- **Comprehensive Plagiarism Detection**: Heimdallr scans through a vast array of sources, including web content, books, papers, photographs, and previous student works, leaving no room for undetected plagiarism.
- **Precise Location Identification**: It doesn't just flag potential instances of plagiarism; Heimdallr provides pinpoint accuracy by highlighting the exact lines or words in question, enabling users to swiftly address the issue.
- **Source Attribution**: The system not only detects plagiarism but also identifies the source that is being plagiarized.

Method

Heimdallr was built following an event-oriented architecture with support for domain modeling. Presenting a clean architecture, allowing changes in each layer has minimal effect on others. The arrangement of this division is easily identified in the package naming:

1. Domain – the core business objects.
2. Adapters – I/O abstractions.
3. Service – Event handling orchestration
4. Entry point –application drivers.

Domain Modeling

The problem that the application tries to solve is plagiarism in students' assignments, being the main entity modeled in an Assignment data class. This record represents persistent writings, saved in a convenient format that facilitates its identification, and comparison.

```
class Assignment (BaseDocument):
    title: str
    author: str
    content: list[str]
    date: datetime.date | None
    similarities: list[AssignmentVerification] | None
```

Notice the use of type annotations offer better legibility. This model will be persisted in a database. The BaseDocument superclass contains an auto-generated identifier using UUID4 format.

Driven Adapters

Inward-facing adapters present a layer of abstraction for reading, and writing operations performed over a database and files.

An `AssignmentReader` interface is responsible for mapping text into a domain model. With only one named method, guarantees, no matter what implementation is done, that a persistent entity can be obtained.

```
class AssignmentReader (abc.ABC):
    @abc.abstractmethod
    def read(file: UploadFile) -> Assignment:
        raise NotImpementedError
```

The current interface implementation utilizes spaCy for sentence recognition: an Assignment content consists of a list of sentences. Also, this NLP library helps with author recognition using tokenization and token type for each sentence. For each different content type supported, a dependency is needed. PymuPDF, pytextract and docx2txt are used for text processing.

For convenience and familiarity, the database chosen was MongoDB. With Motor engine, non-blocking I/O operations are performed. Not delving into implementation details, a repository pattern is used for basic CRUD operations.

Event-Driven Orchestration

An orchestration layer is needed for executing our service value: checking plagiarism in writings. Services receive an input and produce a value. These inputs are orders originated from another source, it could be another application service, or an external source. Represented by data classes, **commands**. Commands expect the system to produce an effect, something must happen. Once the system reacts to a command, a notification is produced, an **event**. This is visible in the service interface.

```
class AssignmentVerifier (abc.ABC):
    @abc.abstractmethod
    async def veriify(command: VerifyAssignment) -> AssignmentVerified:
        raise NotImpementedError
```

Notice the naming of the data classes. Commands are named after in imperative mood, clearly representing an order. Meanwhile, events are named in past tense, something that has happened.

As an orchestrator, the implementation makes use of the aforementioned abstractions for file mapping, and database operations.

The code block below is not the final implementation but a high-level example of how a service can be developed dependent on adapters. It uses *spaCy's* implementation of cosine similarity for each sentence.

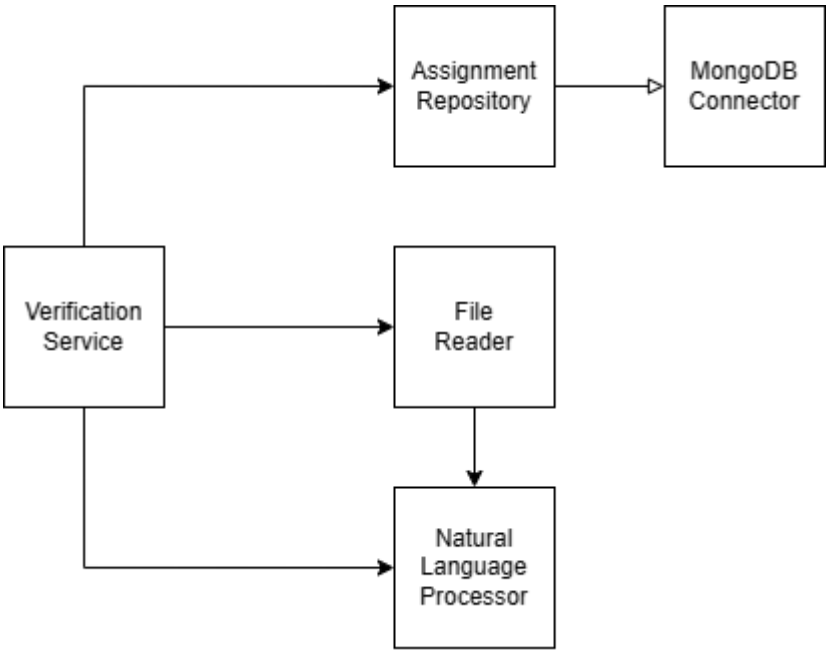

Figure 1 – Assignment Verification Service

```
async def verify(self, command: VerifyAssignment) -> AssignmentVerified:

    # use the AssignmentReader adapter to obtain a domain object
    entry = self.reader.read(file=command.file)
    entry.id = command.id

    # use the Repository/DB adapter to obtain all previous writings
    assignments: list[Assignment] = await self.repository.find_all()
    comparisons: list[AssignmentCompared] = []

    # compare entries with each stored assignments
    for assignment in assignments:
      # compare assignments compares each sentence
      result = self.compare_assignments(assignment, entry)
      if result.similarities:
        comparisons.append(result)

    # map the comparison results to the AssignmentVerification model
    verifications = [AssignmentVerification(**comparison.model_dump())
                     for comparison in comparisons]
    entry.similarities = verifications

    # use the DB adapters to persist the operation
    persisted = await self.repository.save(entry)

    # notifies the system an event has occurred
    return AssignmentVerified(
        id=persisted.id,
        title=persisted.title,
        author=persisted.author,
        similarities=comparisons,
    )


def compare_sentence(
 self,
 sentence: str,
 entry_sentence: str) -> SentenceCompared:
    persisted_sentence_doc = self.nlp(sentence)
    entry_sentence_doc = self.nlp(entry_sentence)
    # cosine similarity percentage
    similarity = persisted_sentence_doc.similarity(entry_sentence_doc)

    return SentenceCompared(
        present=sentence,
        compared=entry_sentence,
        plagiarism=similarity)
```

Plagiarism detection as a Service

The added value of Heimdallr is providing NLP evaluation through an API. For that reason, FastAPI microframework was adopted for developing outward-facing adapters, application's drivers.

FastAPI offers high performance, high development speed, easy and intuitive lines of code, and is standards-based. It follows OpenAPI [1] and JSON Schema[2].

In the auto-generated documentation page, we can distinguish different path groups.

1. Monitor – used for checking application availability.
2. Queries – request for information about a system. Does not make any changes.
3. Commands - request for a specific action to occur in the system.

The UI provides examples of both request and response, and it also describes the used schemas. These entry points may reutilize both service and adapter abstractions to perform requested operations.

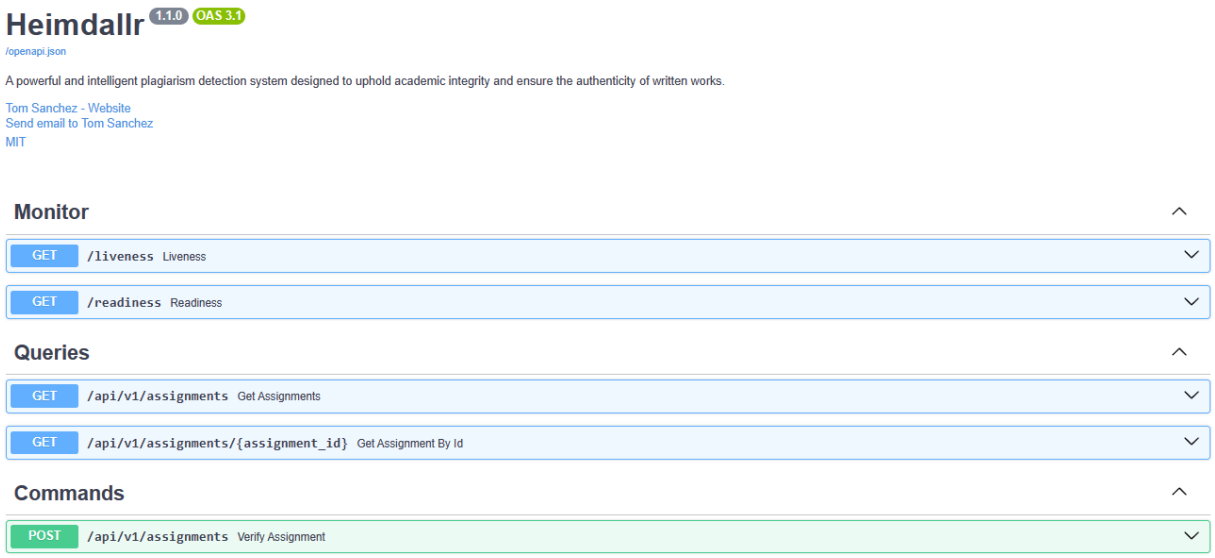

Figure 2 OpenAPI UI

[1] The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for HTTP APIs.

[2] JSON Schema is the vocabulary that enables JSON data consistency, validity, and interoperability at scale.

Results

Performance issues

The idea was to verify each sentence of a new entry against all other sentences of all stored assignment. As response time started to increase, solutions to needed to start popping up.

At first, it was tried to be fixed as an asynchronous endpoint: when a file is submitted, a unique UUID is returned. With this UUID, the assignment can be later retrieved to see its plagiarism results. This solved client timing out waiting for server response issue. However, processing time was still too expensive.

Another idea was to only compare long sentences. It was researched that on average, Spanish has 5.6 average letters per word[3], and arbitrarily decided that using 3 words with less characters than the average isn't being plagiarized. This had little to zero impact. Comparing one assignment with only 14 others will take up to 15 minutes.

Finally, a prior discard idea came up. Comparing whole documents, not sentence by sentence. For that, a threshold needed to be defined. Known assignments with the same requirements were manually compared. Using "Wikinomics" assignment. When evaluating Melanie Blejter's work with a threshold of 0.99.

Author	Same Assignment	Similarity	Seconds
Jourdan Martin	YES	0.9929278646150054	71.27109
Marcelo Brossi	YES	0.9935113489761744	90.099331
Ivo Ursino	NO	0.9901813848392802	89.807355

It was seen that unrelated assignments were also having similarity higher than 0.99. Having updated threshold to 0.991 Testing again with Santiago Peralta's take on "Wikinomics".

Author	Same Assignment	Similarity	Seconds
Jourdan Martin	YES	0.9922299777151997	71.809523
Marcelo Brossi	YES	0.991994420219641	89.406367
Melanie Blejter	YES	0.9956091815513386	90.936489

Without changing anything, for Miguel Arteaga Orozco's writing.

Author	Same Assignment	Similarity	Seconds
Santiago Peralta	YES	0.9941490039858188	104.084537
Marcelo Brossi	YES	0.991522754728401	114.398007
Melanie Blejter	YES	0.9952789647155554	111.424194

There wasn't enough similarity with Jourdan Martins' assignment. Manually checking it, it has 0.9896672681699993.

A threshold of 0.991 was considered acceptable.

[3] See also: Why does content expand when translated? – on inter contact translations' blog.

Topic classification

Another challenge was to categorize assignments. Having manually labeled 100 assignments with different topics, a *train test split* with a size of 0.33 and random state of 42 was done for text classification.

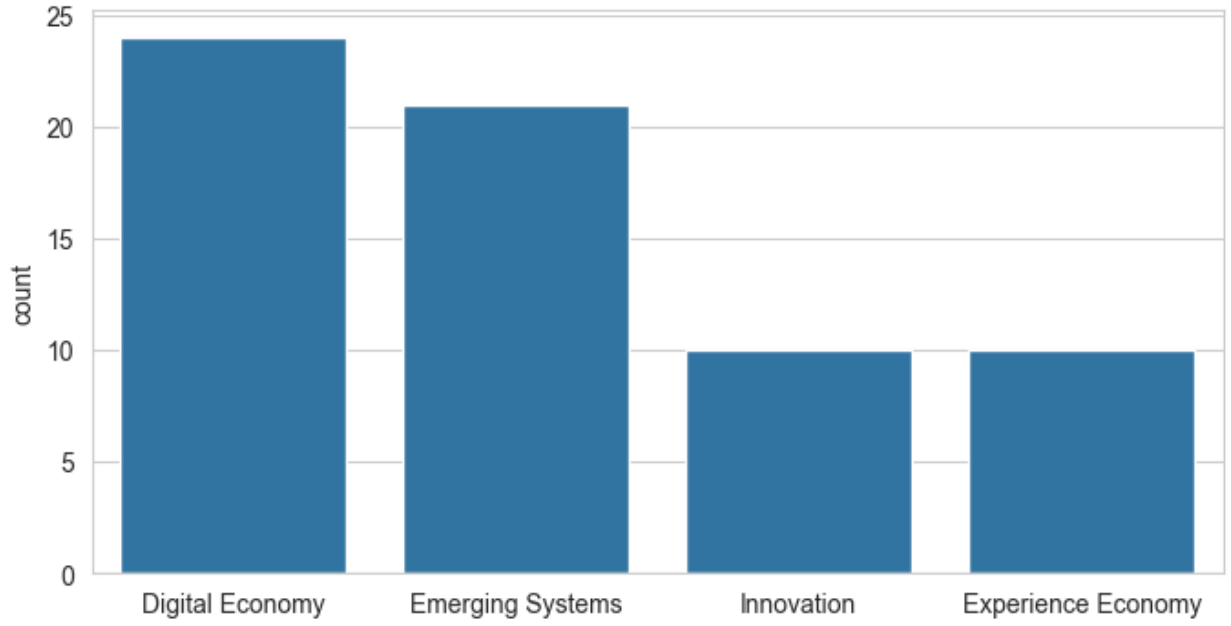

Figure 3 Test Split by Topic

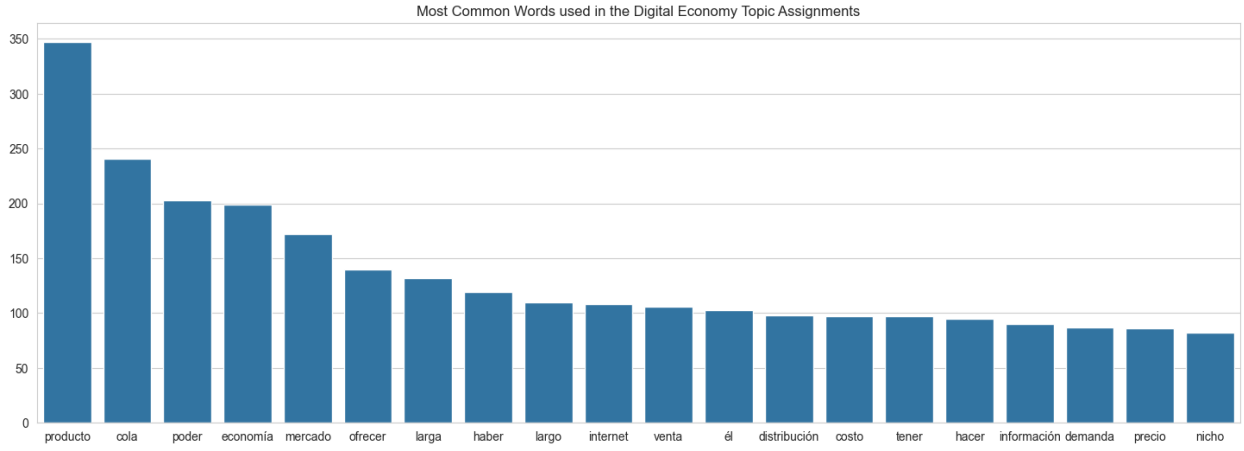

Figure 4 Most Common Words in Digital Economy

More graphs can be seen on the included appendix.

It is seen that "producto" and "cola" are in the top words for "Digital Economy". It is obvious that "The Long Tail" assignment was labeled as "Digital Economy".

The top words for "Innovation" are "adopcion" and "producto".

Metrics Analysis

accuracy: 0.9696969696969697

Top 10 features used to predict.

Class 1 best:

(-0.029102985829993 86, 'ejemplo')

(-0.019688603502727383, 'e.')

(-0.016916017642919 71, 'poder')

(-0.016446214639700 39, 'individuo')

(-0.015634058315695205, 'sistema')

(-0.014470155004150373, 'idea')

(-0.014373718553172994, 'cliente')

(-0.014351650689569237, 'forma')

(-0.013436612415077655, 'preguntar')

(-0.013324425487789 07, 'desarrollar')

Class 2 best:

(0.042387846807027 82, 'economía')

(0.029182893807513717, 'digital')

(0.027224404983200518, 'fuente')

(0.020542679564955 03, 'energía')

(0.019831701811292908, 'internet')

(0.017813780608358233, 'cola')

(0.015380094727189587, 'largo')

(0.015285828149245853, 'red')

(0.012455618860655 32, 'bien')

(0.012392980115877975, 'cualquiera')

	Precision	Recall	f-1 score	support
Emerging Systems	1.00	0.92	0.96	12
Experience Economy	1.00	1.00	1.00	7
Digital Economy	1.00	1.00	1.00	5
Innovation	0.90	1.00	0.95	9
Accuracy			0.97	33
Macro avg	0.97	0.98	0.98	33
Weight avg	0.97	0.97	0.97	33

Topic similarities

Using 98 manually labelled assignments.

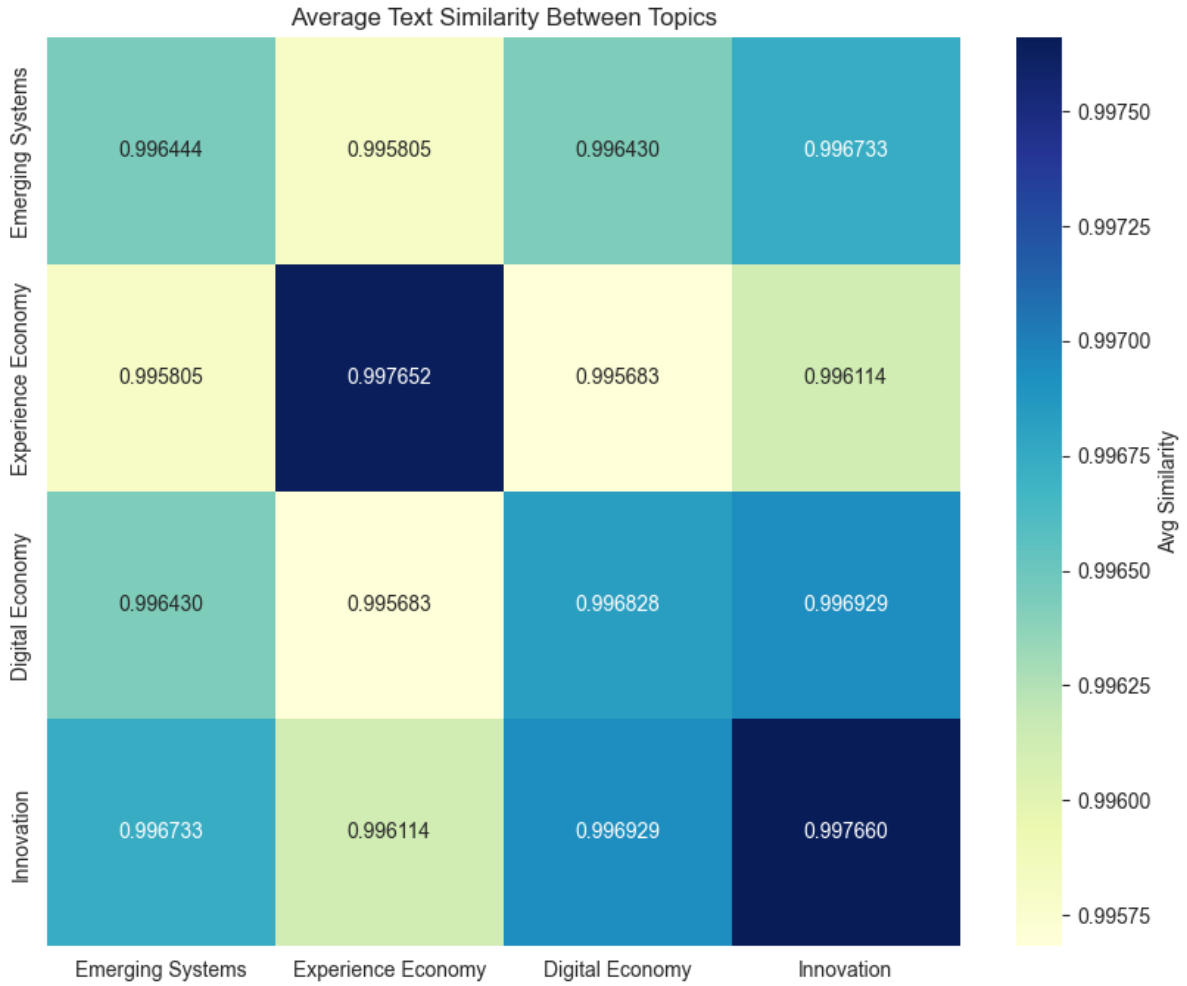

Figure 5 Similarity Heat map

	Emerging Systems	Experience Economy	Digital Economy	Innovation
Emerging Systems	0.996444	0.995805	0.99643	0.996733
Experience Economy	0.995805	0.997652	0.995683	0.996114
Digital Economy	0.99643	0.995683	0.996828	0.996929
Innovation	0.996733	0.996114	0.996929	0.99766

$$avg(similarity) = 0.997146$$

It's seen that with a similarity of 0.997 it has high probability of having equal topic.

About this document

License

Author

Tomas Agustin Sanchez. For more information about the author, visit https://tomsanchez.com.ar.

References

Andreas C. Müller, Sarah Guido (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly editions.

Harry J.W Percival, Bob Gregory (2020). Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices. O'Reilly editions.

Matt Hannibal (2015). spaCy. Official documentation website.

Susan Li (2018). Machine Learning for Text Classification Using SpaCy in Python. Towards Data Science Medium publication.

Sebastián Ramírez Montaño (2023). FastAPI. Official documentation website.

APENDIX A

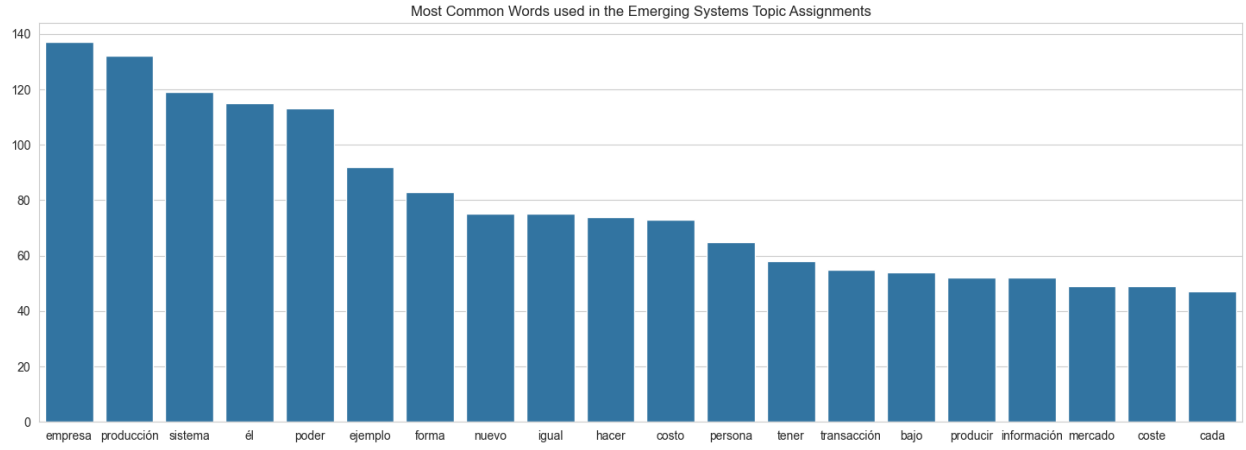

Figure 6 Most common words in Emerging Systems

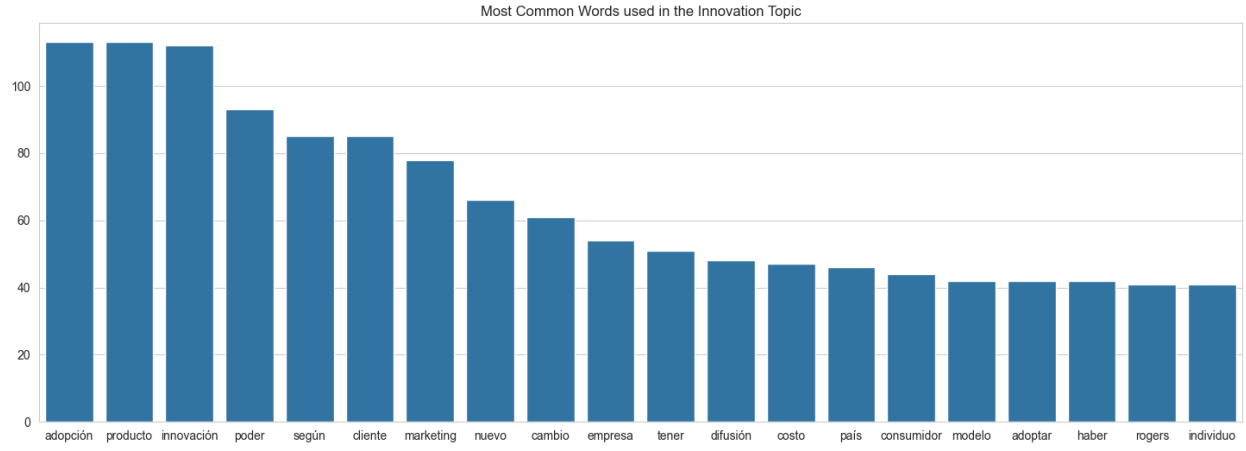

Figure 7 Most common words in Innovation

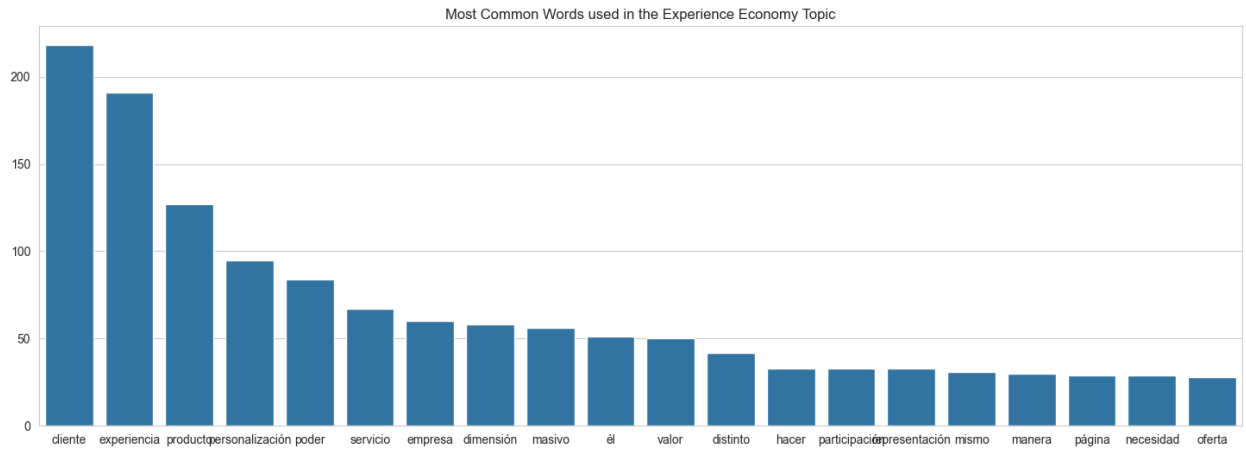

Figure 8 Most common words in Experience Economy