Contents

List of Figures

List of Tables

List of acronyms

- ORM = Object Relational Mapping - IDE = integrated development environment

- UML = Unified Modeling Language

- AI = Artificial intelligence

- ML = machine learning

- NLP = Natural language processing

- NLTK = Natural Language Toolkit

General Introduction

WE delved into the implementation of machine learning techniques, carefully selecting the most suitable model from a range of options to add an extra layer of security to our platform.

Sentiment analysis of social network tweets is a powerful technique for understanding users' emotions and opinions. Using natural language processing methods and sentiment analysis libraries, it is possible to transform large amounts of textual data into actionable information for a variety of applications.

Chapter 1

Tweets sentiment analysis

1.1 Introduction

In today's digital world, ensuring the security and privacy of user information is critical. We have integrated the power of artificial intelligence and machine learning into our application as part of our commitment to providing a robust and reliable matching system. The goal is to improve the accuracy

1.1.1 The utility of Natural Language Processing

Our application analyzes the semantic similarity between the user's stored security question and the answer provided during password changes using advanced natural language processing techniques and machine learning algorithms. We can use AI to assess the level of matching in a more intelligent and accurate manner.

The use of AI and ML models allows our system to learn from patterns and behaviors, gradually improving its matching capabilities. Our application can detect subtle linguistic nuances, account for variations in language usage, and adapt to evolving user responses by utilizing cutting-edge algorithms.

We will delve into the technical aspects of our implementation in the following sections of this report, including the AI and ML models used, data preprocessing techniques used, and evaluation metrics used to assess the performance of our matching system. In addition, we will discuss the potential benefits and challenges of incorporating AI into our application.

1.1.1.1 How does a computer understand natural language?

NLP is a branch of computer science that studies how computers interact with human natural languages. Speech recognition, natural language understanding, natural language generation, and machine translation are all sub-fields of NLP.

NLP is a difficult field to study because human language is complex and nuanced, on the other hand, it has the potential to transform the way we interact with computers. NLP, for example, can be used to develop chat-bots that can converse with humans or systems that can automatically translate text from one language to another [6].

NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment.

1.1.1.2 Utility of natural language processing

Here are some real-world examples of the utility of Natural language processing [6]:

- **Speech recognition**: Natural language processing is used in speech recognition systems to convert spoken language into text. This technology can be found in a wide range of applications, including voice assistants, dictation software, and call centers.
- **Natural language understanding** : NLP is used to extract meaning from text in natural language understanding systems. Search engines, machine translation, and question answering systems all use this technology.
- **Natural language generation** : NLP is used in natural language generation systems to generate text that humans can understand. Email spam filters, chat bots, and text-to-speech systems are all examples of how this technology is used.
- **Machine translation**: Natural language processing is used in machine translation systems to translate text from one language to another. This technology is used in a wide range of applications, including online translation services, multilingual websites, and international trade.

1.1.1.3 What is the matching process?

Matching is the process of finding two or more items that are similar or related to each other. Matching can be used in a variety of applications, such as:

- **Search engine**s: In search engines, matching is used to find documents that are relevant to a user's query.
- **Machine translation**: In machine translation, matching is used to find the correct translation for a word or phrase.
- **Question answering systems**: In question answering systems, matching is used to find the answer to a question.
- **Recommendation systems**: In recommendation systems, matching is used to recommend products or services to users.
- **Fraud detection**: In fraud detection, matching is used to identify fraudulent transactions.

Matching also can be performed using a variety of techniques, such as:

- **Text similarity**: Text similarity measures the similarity of two pieces of text.

- **Feature similarity**: The similarity of two sets of features is measured using feature similarity.
- **Pattern matching** :which is a technique for detecting patterns in data.
- **Machine learning**: Machine learning can be used to learn data patterns and then apply those patterns to new data.

Matching can be used to add an extra layer of security to our system by detecting the level of similarity between the security question response provided by the user, stored in the database, and the response while changing password.

This can be done by using a matching algorithm to compare the two responses. If the two responses are similar, then the user is likely to be the same person who created the account. However, if the two responses are not similar, then the user may be trying to gain unauthorized access to the account.

It is important to note that users may forget or change their security question responses semantically over time. A capable system should be able to handle these kinds of circumstances gracefully.

1.1.2 AI models for matching and NLP

We'll talk about the various AI models that can be used for matching and natural language processing :

- **Chat Generative Pre-Trained Transformer model**: Chat Generative Pre-Trained Transformer [1] is a large language model developed by OpenAI. It is trained on a massive dataset of text and code, and it can generate text, translate languages, write different kinds of creative content, and answer your questions in an informative way. the model is still under development, but it has already learned to perform many kinds of tasks, including Following instructions and completing requests thoughtfully. Using its knowledge to answer your questions in a comprehensive and informative way, even if they are open ended, challenging, or strange. Generating different creative text formats of text content, like poems, code, scripts, musical pieces, email, letters and more. It will try its best to fulfill all your requirements.
- **Doc2Vec**:Doc2vec is a machine learning model that creates vector representations of documents. These vector representations can be used for a variety of tasks, such as document classification, sentiment analysis, and question answering. Doc2vec is trained using a neural network. The neural network is given a set of documents,

Model	*Positive aspects*	*Drawbacks*
Doc2Vec	*Wide range of features for working with text *Can be used for a variety of NLP tasks, such as text classification, sentiment analysis. *Open source and free to use	*Not as accurate as some other models
Word2Vec	*Easy to use *Can be effective for a variety of tasks *Is open source and free to use	*Can be less accurate than other models *Not as well-suited for large-scale projects
BERT	*Very accurate *Can be used for a variety of tasks like answering and text summarization *Is open source and free to use	*Requires a large amount of data to train *Can be slow for some tasks
LLMS	*Can generate human-quality text *Can be used for a variety of tasks	*Can be biased or offensive *It is important to use it responsibly *Paid version only

Table 1.1: Positive aspects Drawbacks of the different AI models

and it learns to predict the next word in each document. The neural network also learns to predict the document that a given word belongs to [5].

Once the neural network is trained, it can be used to create vector representations of documents. These vector representations can be used for a variety of tasks. For example, a document classification model can be trained to classify documents into different categories based on their vector representations [5].

- **Bidirectional Encoder Representations from Transformers**: BERT is a transformer-based model that has demonstrated cutting-edge performance in a variety of NLP tasks, including sentence similarity. BERT can generate contextualized word representations by taking into account both left and right context, allowing it to better capture the meaning and nuances of sentences [3] .

- **Word2Vec**: Word2Vec [8] is a popular word embedding model. It captures semantic relationships between words by representing them as dense vectors in a continuous vector space. A representation for the entire sentence and its similarity to other sentences can be obtained by averaging the word vectors of the words in a sentence, Word2Vec can make strong estimates about a word's meaning based on their occurrences in the text. These estimates yield word associations with other words in the corpus .

The table 1.1 compares the various AI models and their benefits and drawbacks, In order to select the best AI model for our specific task. It shows also that each AI model has its own set of advantages and disadvantages. Doc2Vec, for example, is fast and efficient, making it a good choice for large-scale projects , open source and free to use, which makes it a cost-effective option. But it can be difficult to learn how to use. Word2Vec is simple to use, but it is not as accurate as other models. BERT is very accurate, but it takes a lot of data to train. ChatGPT can generate text that is human-quality, but it can also be biased or offensive.

1.1.3 Libraries for matching and NLP

We'll talk about the various libraries that can be used for matching and natural language processing :

- **Gensim**: "Generate Similar" is a popular open source natural language processing library used for unsupervised topic modeling [4]. It uses top academic models and modern statistical machine learning to perform various complex tasks such as Building document or word vectors, Corpora, performing topic identification, performing document comparison (retrieving semantically similar documents), analysing plain-text documents for semantic structure [4].
- **NLTK**: NLTK [7] is a Python library for natural language processing. It is used for tasks such as text classification, tokenization, stemming, and parsing. NLTK provides a variety of tools and resources for natural language processing, including corpora, datasets, and tutorials. NLTK is designed for a variety of tasks, including text classification, tokenization, stemming, and parsing.

These are just a few of the many NLP and matching libraries available. The best library for a specific task will be determined by the task's specific requirements.

Aside from these libraries, a variety of commercial NLP and matching solutions are available. These solutions are typically more comprehensive in terms of features and functionality than open-source libraries, but they are also more expensive.

Conclusion

The main difference between gensim, and nltk is the type of tasks they are designed for. Gensim is designed for tasks such as topic modeling, document similarity, and text classification. NLTK is designed for a variety of tasks, including text classification, tokenization, stemming, and parsing. Gensim is a powerful natural language processing tool. It can be used to build and train word embedding models, which can then be used to perform tasks like text classification, sentiment analysis, and named entity recognition. Gensim is ideal for my job because it is quick, efficient, and simple to use. It can be used to quickly and accurately process large amounts of text. Furthermore, Gensim is open source and free to use, making it a cost-effective option.

Criteria	**Word2Vec**	**Bert**	**Doc2Vec**
Model Architecture	Neural	Transformer-based	Neural
Pre-training	Pre-trained and Fine-tuning	Pre-trained and Fine-tuning	Pre-trained
Complexity	Moderate	Large	Moderate
Generalization	Moderate	Good	Good
Training Data	Large labeled data	Large labeled data	Large labeled data
Language Support	Language-dependent	Multilingual	Multilingual
Accessibility	Open-source	Open-source	Open-source
Documentation	Good	Moderate	Good

Table 1.2: Comparative table of the different AI models

1.1.3.1 Discussion

When choosing an NLP or matching library, it is important to consider the following factors:

- The specific tasks that need to be performed
- The size and complexity of the data sets
- The budget
- The level of support that is required

The table 1.2 presents a comparison of different NLP models, highlighting their key characteristics and attributes. The models included in the table are Word2Vec, BERT, and Doc2Vec. Each model is evaluated based on several criteria.

Doc2Vec is a powerful tool for learning document embeddings that capture the semantic meaning of text provided by the gensim library. And here are the reasons:

- **Simplicity and Ease of Use:** Doc2Vec in the gensim library has a simple, easy implementation and is relatively simple to use. It offers a simple API for training document embeddings, allowing you to concentrate on your specific tasks rather than worrying about complex details.
- **Training Efficiency:** Doc2Vec training is effective and can handle large document corpora. It learns document representations using the distributed memory and distributed bag of words approaches, which can capture both local and global context information.
- **Open-Source and Community Support:** The gensim library, which includes the Doc2Vec implementation, is open-source and has a vibrant community. Documentation, code examples, and community support are available, making it easier to understand, implement, and troubleshoot.

However, there might be reasons to consider eliminating Chat Generative Pre-Trained Transformer model due to it being a paid service, GPT-3.5 is a commercial service and requires payment to access and use the model, opting for a free or open-source alternative like Doc2Vec can be a more economical choice. As for BERT which is a powerful language model, but its complexity might be a factor in deciding to eliminate it, since it is a large and complex model that typically requires significant computational resources for training and inference. BERT often requires large amounts of labeled data and it might be challenging to fully leverage the full potential of the model .Implementing and fine-tuning BERT can be more involved compared to simpler models like Doc2Vec.

Given these considerations, choosing a simpler and more lightweight model, such as Doc2Vec from the gensim library, can be a practical choice, providing efficiency, ease of use, versatility, and open-source benefits for a variety of document-based NLP tasks.

1.1.4 Gensim library

Gensim “Generate Similar” can be used to identify latent topics in a corpus of text. This can be used for tasks such as clustering documents, summarizing text , it is a robust natural language processing and machine learning tool.

1.1.4.1 Gensim Various Features

It has a diverse set of features and capabilities that can be applied to a variety of tasks [4] :

- **Scalability**: Using its incremental online training algorithms, Gensim can easily process large and web-scale corpora. It is scalable because the entire input corpus does not have to reside entirely in Random Access Memory (RAM) at any given time. In other words, regardless of the size of the corpus, all of its algorithms are memory-independent.
- **Robust**: Gensim is a robust system that has been used in various systems by various people and organizations for over 4 years. We can quickly add our own input corpus or data stream. It is also very simple to extend with additional Vector Space Algorithms.
- **Platform independence**: As we all know, Python is a very versatile language, and Gensim, being pure Python, runs on all platforms that support Python and Numpy (such as Windows, Mac OS, and Linux).
- **Open source and a thriving community of support**: Gensim is licensed under the OSI-approved GNU LGPL license, which allows it to be used for free for

both personal and commercial purposes. Any changes made to Gensim are open-sourced and receive a lot of community support.

1.1.4.2 Code Dependencies

Gensim 3.8.0, which was released in July 2019 [4], is the most recent version that should work on any platform that has Python 2.7 or 3.5+ and NumPy installed. It is actually determined by the following software :

- **Python**: Gensim is tested with Python versions 2.7, 3.5, 3.6, and 3.7.
- **Numpy** : NumPy is a package for scientific computing with Python. It can also be used as an efficient multi-dimensional container of generic data.
- **smart-open** : It is a Python 2 and Python 3 library that is used to stream very large files efficiently. Gensim relies on the smart-open Python library to open files on remote storage as well as compressed files transparently.

1.1.4.3 Core Concepts of Gensim

Following are the core concepts and terms that are needed to understand and use Gensim:

- **Document**:It refers to some text.
- **Corpus**: It refers to a collection of documents.
- **Vector** : Mathematical representation of a document is called vector.
- **Model**: It refers to an algorithm used for transforming vectors from one representation to another.

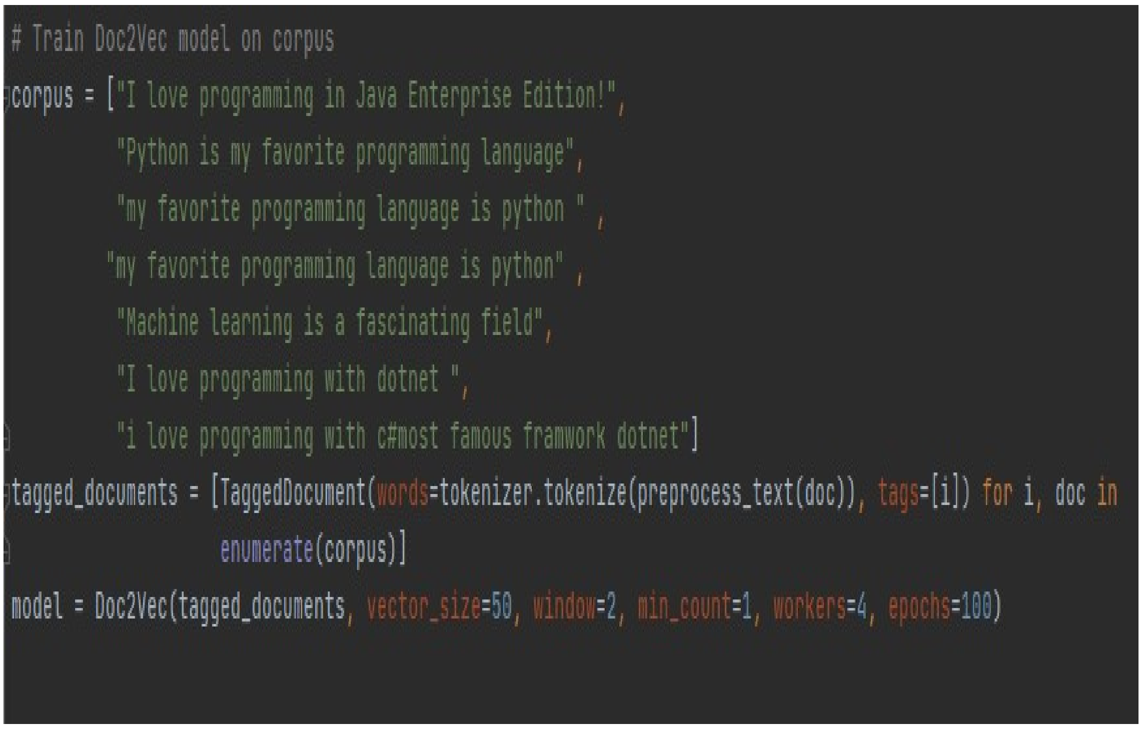
What is a Corpus?
A corpus may be defined as the large and structured set of machine-readable texts produced in a natural communicative setting. In Gensim, a collection of document object is called corpus.

A corpus in Gensim serves the following two roles:

- **Serves as input for training a model**:The very first and important role a corpus plays in Gensim, is as an input for training a model. In order to initialize model's internal parameters, during training, the model look for some common themes and topics from the training corpus. As discussed above, Gensim focuses on unsupervised models, hence it doesn't require any kind of human intervention.

- **Serves as topic extractor** : Once the model is trained, it can be used to extract topics from the new documents. Here, the new documents are the ones that are not used in the training phase.

The figure 1.1 show us an example of a small corpus used in our matching algorithm which contains 7 documents. Here, every document is a string consisting of a single sentence.



```
# Train Doc2Vec model on corpus
corpus = ["I love programming in Java Enterprise Edition!",
          "Python is my favorite programming language",
          "my favorite programming language is python " ,
         "my favorite programming language is python" ,
          "Machine learning is a fascinating field",
          "I love programming with dotnet ",
          "i love programming with c#most famous framwork dotnet"]
tagged_documents = [TaggedDocument(words=tokenizer.tokenize(preprocess_text(doc)), tags=[i]) for i, doc in
                   enumerate(corpus)]
model = Doc2Vec(tagged_documents, vector_size=50, window=2, min_count=1, workers=4, epochs=100)
```

Figure 1.1: Example of Corpus

What is a Vector?

What if we want to infer the latent structure in our corpus? For this, we need to represent the documents in a such a way that we can manipulate the same mathematically. One popular kind of representation is to represent every document of corpus as a vector of features. That's why we can say that vector is a mathematical convenient representation of a document.

Such vector representation is known as a dense vector, The representation can be a simple like (0, 2), Such sequence is the vector for our document.

Another popular kind of representation is the bag-of-word (BoW) model. In this approach, each document is basically represented by a vector containing the frequency count of every word in the dictionary. The figure 1.2 show us an example of a bag-of-word .

Suppose we have a dictionary that contains the words ['love', 'adore', 'prefer', 'c#',

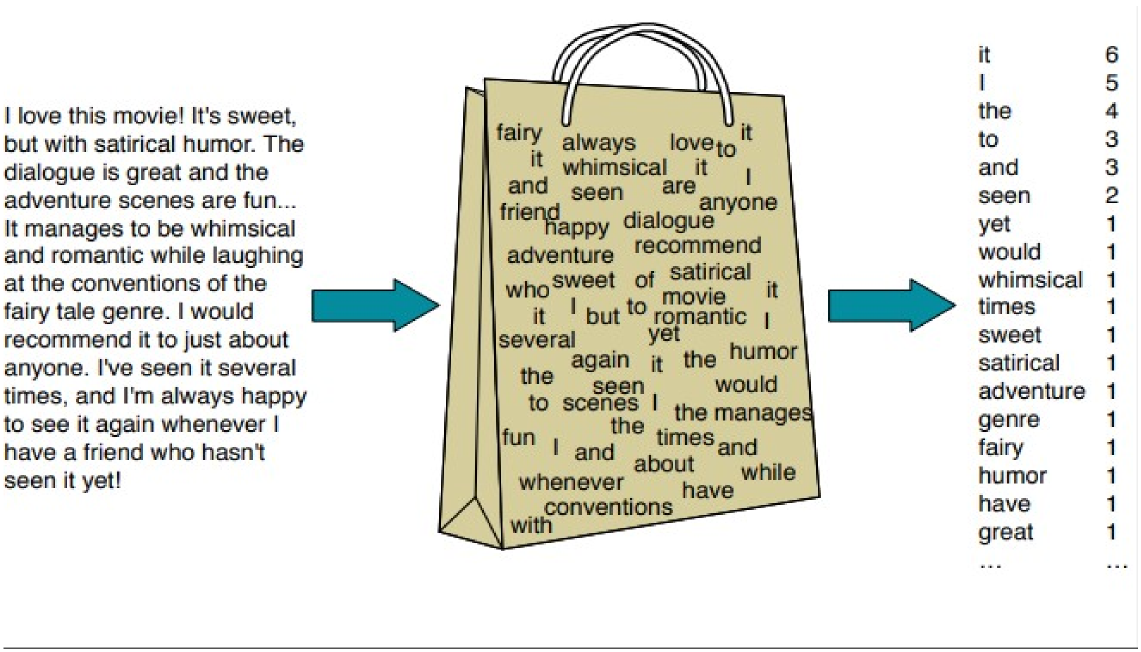

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!
it 6
I 5
the 4
to 3
and 3
seen 2
yet 1
would 1
whimsical 1
times 1
sweet 1
satirical 1
adventure 1
genre 1
fairy 1
humor 1
have 1
great 1
... ...

Figure 1.2: Example of a Bag Of Word
[2]

'JEE']. A document consisting of the string "i love c# and also prefer c# more " would then be represented by the vector [1, 0, 1, 2,0]. Here, the entries of the vector are in order of the occurrences of "love", "adore", "prefer", "c#", and "JEE".

What is a Model?

Once we have vectorised the corpus, we can transform it using models. Model may be referred to an algorithm used for transforming one document representation to other. As we have discussed, documents, in Gensim, are represented as vectors hence, we can, though model as a transformation between two vector spaces. There is always a training phase where models learn the details of such transformations.

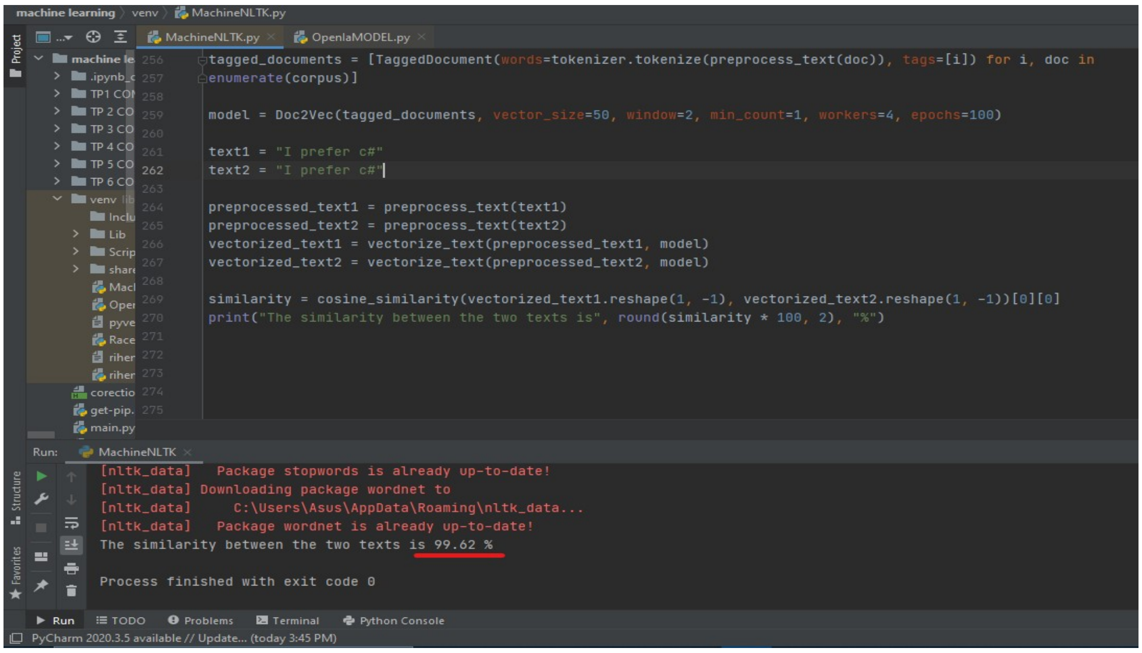
1.1.5 Examples of execution

Cosine similarity can be used in machine learning to compare two text documents. To begin, we must convert each text document into a vector. This can be accomplished using a variety of methods, including doc2vec. Once we have the vectors, we can compute their cosine similarity. The greater the cosine similarity, the greater the similarity between the two text documents.

The cosine similarity formula is defined as follows:

$$\cos(\mathbf{t},\mathbf{e}) = \frac{\mathbf{t}\mathbf{e}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i\mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2}\sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}} \tag{1.1}$$

The figure 1.3 showcases the execution of the code to determine the matching level



```python
tagged_documents = [TaggedDocument(words=tokenizer.tokenize(preprocess_text(doc)), tags=[i]) for i, doc in
enumerate(corpus)]

model = Doc2Vec(tagged_documents, vector_size=50, window=2, min_count=1, workers=4, epochs=100)

text1 = "I prefer c#"
text2 = "I prefer c#"

preprocessed_text1 = preprocess_text(text1)
preprocessed_text2 = preprocess_text(text2)
vectorized_text1 = vectorize_text(preprocessed_text1, model)
vectorized_text2 = vectorize_text(preprocessed_text2, model)

similarity = cosine_similarity(vectorized_text1.reshape(1, -1), vectorized_text2.reshape(1, -1))[0][0]
print("The similarity between the two texts is", round(similarity * 100, 2), "%")
```

```
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Asus\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
The similarity between the two texts is 99.62 %

Process finished with exit code 0
```

Figure 1.3: Matching between two identical texts

between two identical texts using a Doc2Vec model. The texts used for evaluation are :

- **Text 1 :** " I prefer c# "
- **Text 1 :** " I prefer c# "

It demonstrates the successful determination of the model in recognizing the similarity between two identical texts using the Doc2Vec approach.

In the figure 1.4 we are showing the execution of the same algorithm with :

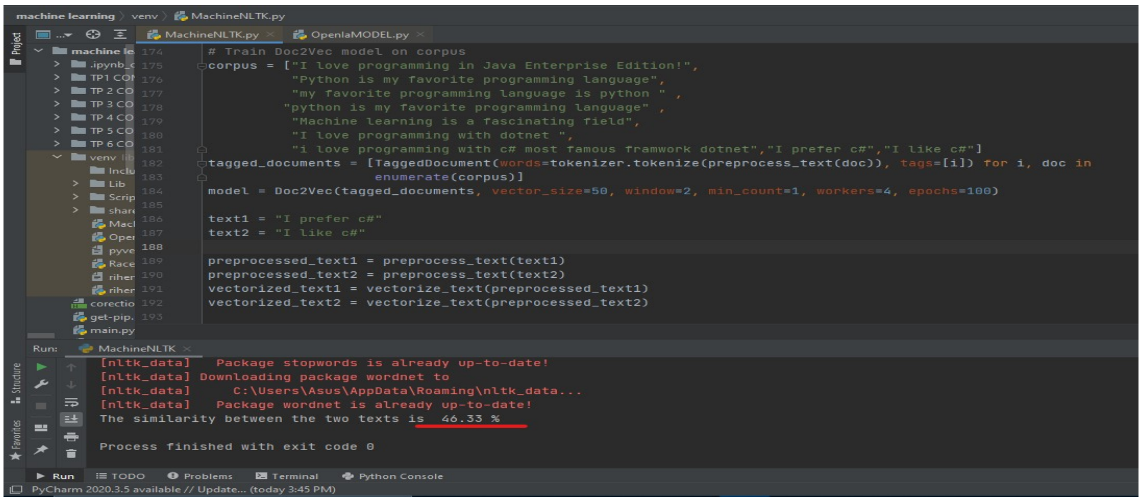
- **Text 1 :** " I prefer c# "
- **Text 1 :** " I like c# "

The code first preprocess the text by removing stop words and lemmatizing the tokens. Then, it vectorizes the text using the Doc2Vec model. Finally, it calculates the cosine similarity between the two vectors.

The model determination was satisfying but can be more accurate. By using a larger corpus of text to train the model, the model can be more accurate.

The figure 1.5 showcases the execution of the code to determine the dissimilarity between two texts using a Doc2Vec model. The texts used for evaluation are representing different preferences:

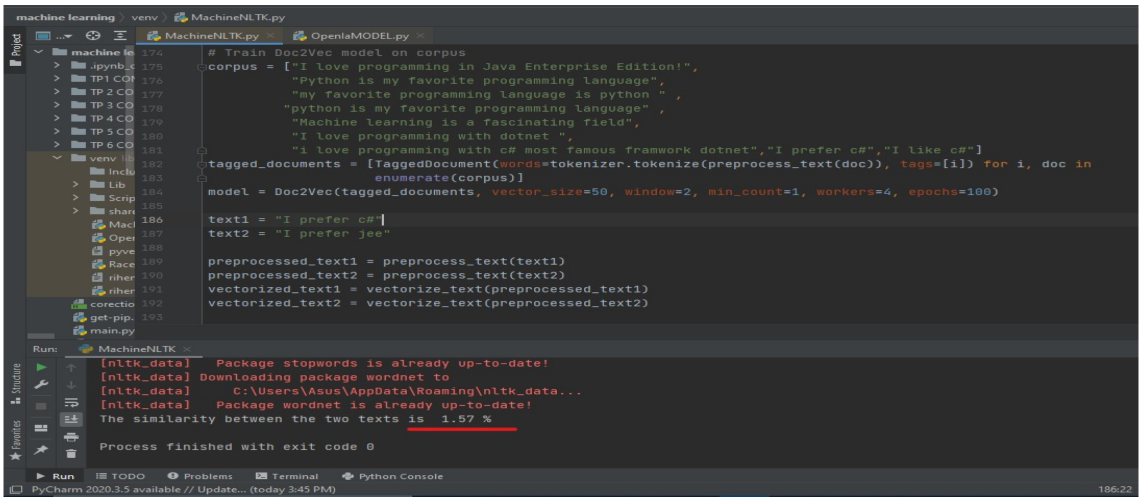
- **Text 1 :** " I prefer c# "



```
# Train Doc2Vec model on corpus
corpus = ["I love programming in Java Enterprise Edition!",
          "Python is my favorite programming language",
          "my favorite programming language is python " ,
         "python is my favorite programming language" ,
          "Machine learning is a fascinating field",
          "I love programming with dotnet ",
          "i love programming with c# most famous framwork dotnet","I prefer c#","I like c#"]
tagged_documents = [TaggedDocument(words=tokenizer.tokenize(preprocess_text(doc)), tags=[i]) for i, doc in
                    enumerate(corpus)]
model = Doc2Vec(tagged_documents, vector_size=50, window=2, min_count=1, workers=4, epochs=100)

text1 = "I prefer c#"
text2 = "I like c#"

preprocessed_text1 = preprocess_text(text1)
preprocessed_text2 = preprocess_text(text2)
vectorized_text1 = vectorize_text(preprocessed_text1)
vectorized_text2 = vectorize_text(preprocessed_text2)
```

```
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Asus\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
The similarity between the two texts is  46.33 %

Process finished with exit code 0
```

Figure 1.4: Matching between two semantically texts



```
# Train Doc2Vec model on corpus
corpus = ["I love programming in Java Enterprise Edition!",
          "Python is my favorite programming language",
          "my favorite programming language is python " ,
         "python is my favorite programming language" ,
          "Machine learning is a fascinating field",
          "I love programming with dotnet ",
          "i love programming with c# most famous framwork dotnet","I prefer c#","I like c#"]
tagged_documents = [TaggedDocument(words=tokenizer.tokenize(preprocess_text(doc)), tags=[i]) for i, doc in
                    enumerate(corpus)]
model = Doc2Vec(tagged_documents, vector_size=50, window=2, min_count=1, workers=4, epochs=100)

text1 = "I prefer c#"
text2 = "I prefer jee"

preprocessed_text1 = preprocess_text(text1)
preprocessed_text2 = preprocess_text(text2)
vectorized_text1 = vectorize_text(preprocessed_text1)
vectorized_text2 = vectorize_text(preprocessed_text2)
```

```
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Asus\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
The similarity between the two texts is  1.57 %

Process finished with exit code 0
```

Figure 1.5: Matching between two different texts

- **Text 1 :** " I prefer JEE "

As the texts have different preferences ("c#" vs. "JEE"), the cosine similarity score will be relatively low and the output of the code indicate the dissimilarity between the texts.

1.2 Conclusion

the integration of artificial intelligence and machine learning techniques has emerged as a powerful approach to augment our understanding of the users' emotions and opinions.

Bibliography

[1] Ben Jones. *ChatGPT Basics: Exploring Its Origins, Uses, and Misuses*. https://dataliteracy.com/product/chatgpt-basics-ebook-pdf/. [Accessed on 2023-05-09] , Data Literacy Press; 1st edition. Apr. 2023.

[2] Daniel Jurafsky and James H. Martin. “Speech and Language Processing”. In: vol. 382. MDM ’05. [Accessed on 2023-05-22]. Stanford University, 2023, pp. 40–200.

[3] M V Koroteev. *BERT: A Review of Applications in Natural Language Processing and Understanding*. https://arxiv.org/abs/2103.11943. [Accessed on 2023-05-08]. Mar. 2021.

[4] Tutorials Point. “Gensim”. In: vol. 283. First’01. [Accessed on 2023-05-06]. Tutorials Point (I) Pvt. Ltd, 2009, pp. 1–38.

[5] Radim Řehůřek. *Doc2vec paragraph embeddings*. https://radimrehurek.com/gensim/intro.html. [Accessed on 2023-05-01]. Dec. 2022.

[6] Anuj Gupta Sowmya Vajjala Bodhisattwa Majumder and Harshit Surana. “Practical Natural Language Processing”. In: vol. 424. A Comprehensive Guide to Building Real-World NLP Systems. [Accessed on 2023-05-02]. O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol., 2020. Chap. Chapter1 [Part1:Foundation], pp. 3–58.

[7] Ewan Klein Steven Bird and Edward Loper. “Natural Language Processing with Python”. In: vol. 499. SE 1999. [Accessed on 2023-05-20]. O’reilly media Inc., 1005 Gravenstein Highway North, Sebastopol., 2009, pp. 17–87.

[8] Vatsal. *Word2Vec Explained*. https://towardsdatascience.com/word2vec-explained-49c52b4ccb71. [Accessed on 2023-04-16]. July 2021.