Horban Hlib, Candidate of Technical Sciences (PhD), Associate Professor, Associate Professor of the Department of Software Engineering, Petro Mohyla Black Sea National University, ORCID: https://orcid.org/0000-0002-6512-3576

Antipova Kateryna, PhD, Associate Professor (acting) of the Department of Software Engineering, Petro Mohyla Black Sea National University, ORCID ID: https://orcid.org/0000-0002-9012-5290

Falenkova Maryna, Senior Lecturer of the Department of Software Engineering, Petro Mohyla Black Sea National University, ORCID ID: https://orcid.org/0000-0001-7797-0142

ARCHITECTURE OF A WEB-BASED SYSTEM FOR ANALYSING THE TONE AND THEMATIC STRUCTURE OF TEXT DATA

Abstract

The article discusses approaches to building a web-based text data analysis system that combines message sentiment classification and thematic modeling within a single software solution. The aim of the study is to develop the architecture and software implementation of a system capable of automating the full text processing cycle: receiving input data, preprocessing, determining emotional tone, identifying key topics, saving results, and presenting them to the user via a web interface.

The methodological foundation consists of natural language processing methods, text vectorization using TF-IDF, sentiment classification using a Support Vector Classifier, and topic modeling based on Latent Dirichlet Allocation.

A modular approach to design was used: the client-side, server API, machine learning module, authentication mechanisms, and data storage were separated. As a result of the research, a system architecture was proposed that ensures the integrated execution of two analytical tasks—determining the text's

sentiment and identifying thematic patterns in the message corpus. The paper describes the main system entities, the mechanism for forming an analysis request, the method of presenting results in JSON format, and the approach to service deployment, in which the base API is implemented in C#, and the analytical processing module is implemented as a separate service on FastAPI.

The scientific novelty of this work lies in the combination of classical machine learning models for text analysis with a distributed service architecture of a web application, which allows separating the computationally intensive natural language processing module from the system's application logic and simplifies scaling, maintenance, and further expansion of functionality.

The practical significance of the results lies in the ability to use the proposed solution for monitoring public sentiment, analyzing user feedback, conducting market research, preprocessing text corpora, and generating analytical reports. The developed system can serve as a foundation for further improvement through the expansion of language support, the adaptation of models to domain-specific corpora, the enhancement of result interpretability, and the implementation of more advanced visualization tools.

Keywords: sentiment analysis, thematic modeling, natural language processing, machine learning, text classification, web system.

Introduction. In today's world, information flows are increasing at a breakneck pace, requiring new approaches to data analysis. One of the promising areas is text sentiment analysis, which provides valuable insights into users' thoughts, emotions, and reactions. Such analysis is useful for various fields, including marketing, sociology, political science, and other areas where it is important to understand public sentiment and reactions to specific events.

Thanks to modern machine learning and natural language processing technologies, it has become possible to automate the process of analyzing texts

based on their content and tone. This not only significantly speeds up the information processing process but also improves the accuracy of the results obtained, which is crucial in the context of analyzing large volumes of data. Topic modeling tools can identify hidden themes and key elements in texts, creating new opportunities for sentiment analysis in a dynamic information space.

Analysis of recent research and publications. Topic modeling is a text analysis method that allows for the detection and identification of topics present in a text dataset [1]. The main idea is that each document can be represented as a combination of different topics with specific probabilities. This text analysis method, combined with sentiment analysis, will allow us to analyze and identify community dynamics.

The main methods of topic modeling include Latent Dirichlet Allocation (LDA), Non-Negative Matrix Factorization (NMF), and modern neural approaches such as BERT Topic Modeling. Each of these methods has its own characteristics and applications.

Latent Dirichlet Allocation (LDA) is a generative model that assumes that each document is a mixture of topics, and each topic is a mixture of words [2]. The main idea behind LDA is to find the topics that best describe the structure of the text data. The model uses a hierarchical approach to identify topics based on word frequencies in documents, allowing it to automatically detect the probability distributions of topics in each document. One of the main advantages of LDA is its ability to process large volumes of text data and identify topics without the need to predefine the number of topics. However, LDA also has its limitations, notably the need to tune hyperparameters, which can significantly affect the results, as well as difficulties in interpreting topics if their number does not correspond to the actual data structure.

Another popular method is Non-Negative Matrix Factorization (NMF), which is also used for topic modeling [3]. NMF treats text data as a term-

document matrix and decomposes this matrix into two non-negative matrices—one for terms-topics and the other for topics-documents. Thus, NMF helps identify latent topics that explain the observed terms in the documents. One of the main advantages of NMF is its ability to identify interpretable topics due to the non-negativity constraint, which provides more understandable and consistent results. However, NMF can be sensitive to initial conditions and requires the prior specification of the number of topics, which can complicate model tuning.

Latent Semantic Analysis (LSA) is another method based on matrix decomposition [4]. LSA uses singular value decomposition of the term-document matrix to reduce dimensionality and extract latent semantic structures in text data. This allows the model to identify synonymic and semantic relationships between words, which helps to better understand the context of topics. The advantage of LSA lies in its ability to reduce noise in the data and identify hidden topics. However, LSA can sometimes generate topics that are difficult to interpret and has limitations in representing contextual relationships based on static matrices.

Modern models, such as BERT (Bidirectional Encoder Representations from Transformers), offer new approaches to topic modeling. BERT uses transformers for deep contextual understanding of text and can be adapted for topic analysis through specific configurations or fine-tuning for topic modeling tasks [5, 6, 7]. The advantage of BERT lies in its ability to account for the context of words on both sides and provide accurate results for complex tasks, but this method requires significant computational resources and a large volume of data for training.

Thus, sentiment analysis and topic modeling methods in machine learning provide various ways to identify and organize topics in text data. The choice of a specific method depends on the specifics of the task, the volume of data, and the desired accuracy of the results.

Problem Statement. The problem addressed in this study is the development of a software system capable of automating the entire cycle of text data analysis: from text ingestion and format validation to the classification of message polarity, thematic structuring, storage of results, and their subsequent presentation to the user. Unlike a standalone ML script, such a system must take into account requirements for access security, user-friendly interface, scalability, maintainability, and integration with external services. Based on this, it was necessary not only to select appropriate algorithms but also to develop a comprehensive architecture for the information system.

Main part. The architectural concept of the system under development is based on a modular division of responsibilities among the client-side, server API, machine learning service, data store, message broker, and authentication server. Several complementary models were used to describe the system's structure: the ERD provides an overview of the database entities and relationships, the DFD depicts information flows and the logic of request processing, and the deployment diagram shows the physical placement of components and the communication channels between them.

The ERD illustrates the relationships between the system's main entities: "User," "Analysis Request," "Sentiment Analysis," "Thematic Modeling," and "Analysis Report" (Fig. 1).

The "User" entity represents a system user who has a unique identifier (uuid), a username, and an account creation date. This is the primary entity to which all text analysis operations are linked.

The "AnalysisRequest" entity is responsible for user requests to analyze text data. Each request has a unique identifier, belongs to a specific user, contains a text corpus for analysis in JSON format, and may be linked to a previous request if it is a continuation or modification of a previously made request. The time when the request was made is also stored.

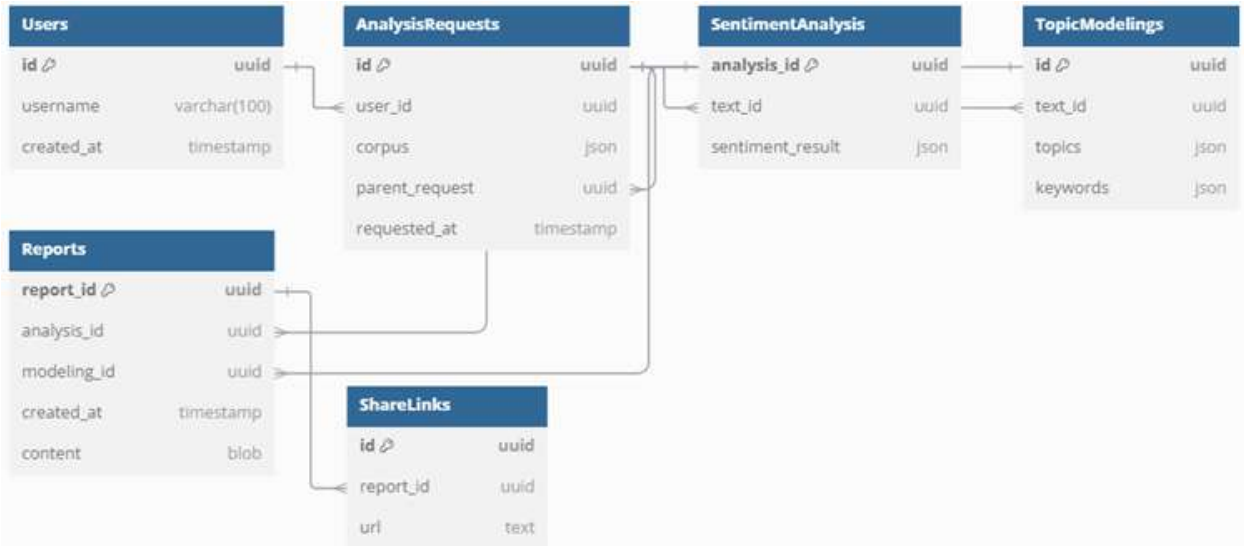


Figure 1 – ERD of a text sentiment analysis system based on thematic modeling

The “SentimentAnalysis” entity represents the results of sentiment analysis for a specific text included in the request. It contains a unique identifier, a link to the text request, and the result in JSON format, which stores data on the emotional tone of the text.

The “TopicModeling” entity stores the results of topic modeling performed on text data. It also has a unique identifier and a link to a specific analysis request. The modeling results store the topics and keywords found during the analysis, presented in JSON format.

The “Report” entity combines the results of sentiment analysis and topic modeling. It contains information about which user created the report, as well as links to the sentiment and topic analysis results. Additionally, the report stores the creation date and the report content in BLOB (binary large object) format, which can be exported.

The “ShareLink” entity allows the user to share generated reports with other users. The link is tied to a specific report and has a short URL for convenient access.

In the context of developing a text sentiment analysis system, the class diagram (Fig. 2) represents the classes responsible for text analysis, data processing, database operations, and report generation.

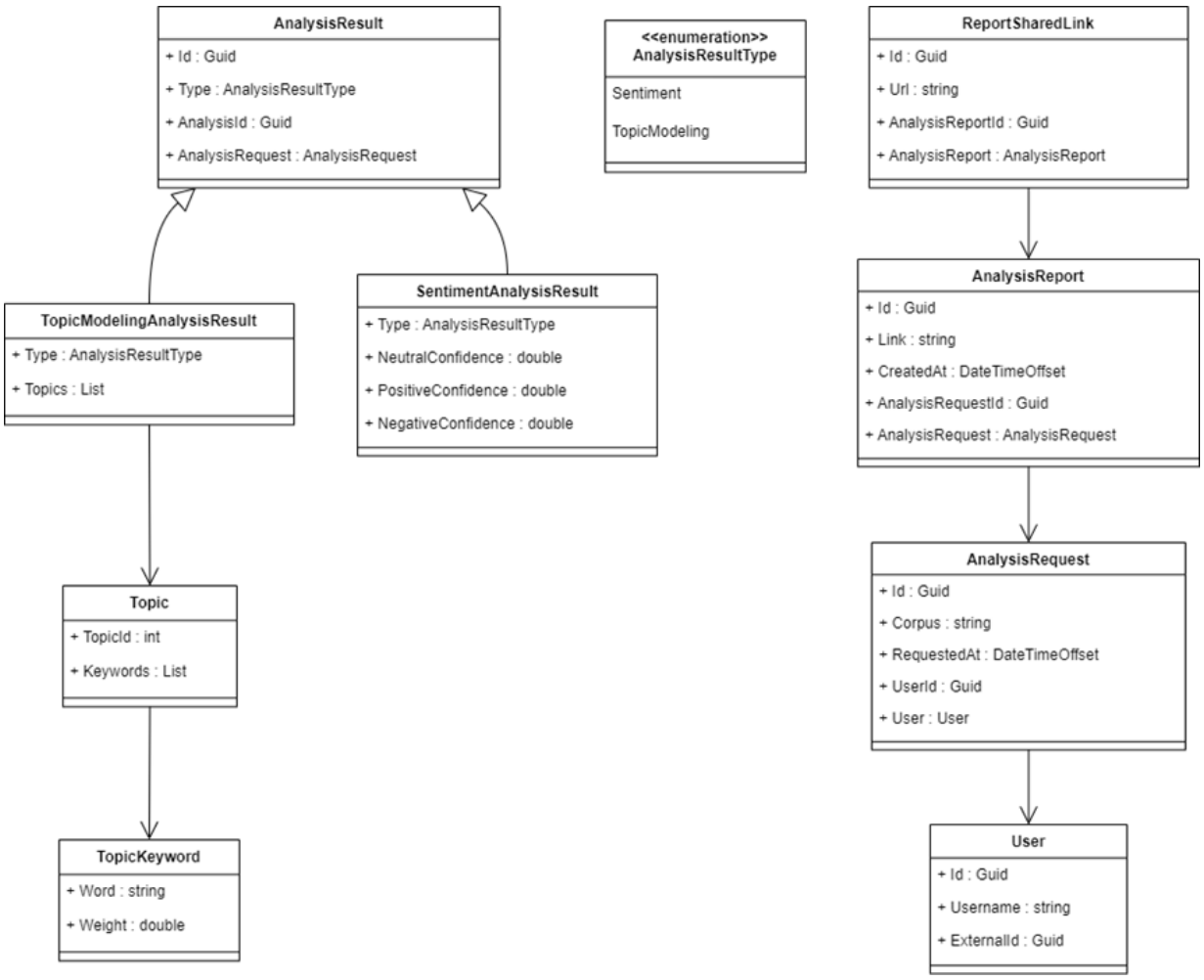


Figure 2 – System Class Diagram

The class diagram consists of the following classes:

– AnalysisReport: represents an analysis report, containing a link and information about the analysis request;

– AnalysisRequest: contains data about the text analysis request, including the text and the user who created it;

– AnalysisResult: represents the analysis result, which can be of the “sentiment analysis” or “thematic modeling” type;

– AnalysisResultType: a list of analysis types available in the system (sentiment analysis or topic modeling);

– ReportSharedLink: contains information about the link for sharing the report;

– SentimentAnalysisResult: the result of sentiment analysis with confidence scores for neutral, positive, and negative sentiments;

– TopicModelingAnalysisResult: the result of topic modeling, containing a list of topics;

– Topic: represents a single topic consisting of keywords;

– TopicKeyword: represents a keyword and its weight for a specific topic;

– User: represents the user who initiated the analysis request.

The system consists of several key components (Fig. 3) that perform their specialized functions. One of them is TextAnalyzer.Core, which acts as a central module with base models and interfaces. It provides fundamental classes and types used in all other system modules. Thanks to this, TextAnalyzer.Core serves as a foundation for expanding functionality and implementing new features.

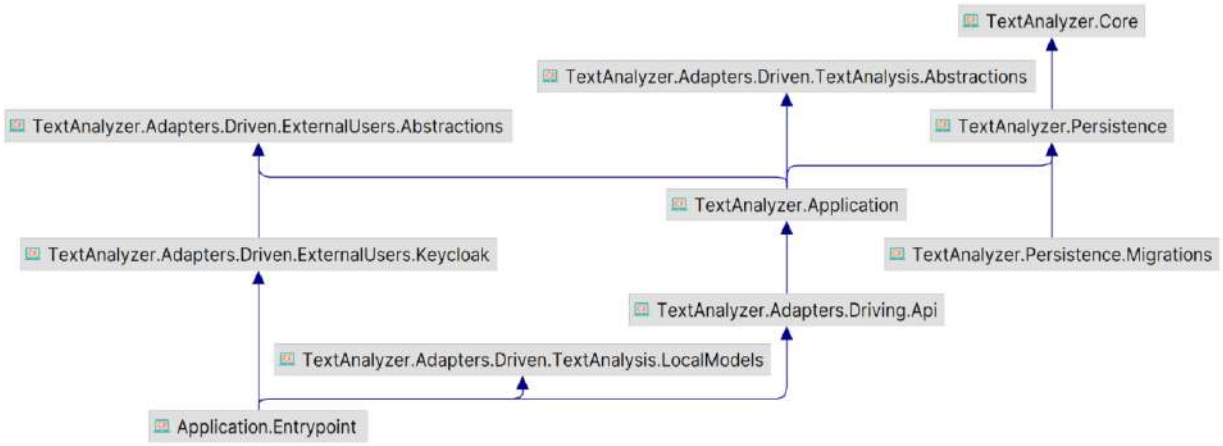


Figure 3 – Project Diagram

Another important module is TextAnalyzer.Application, which implements the system's business logic. It coordinates interaction between various

components, ensuring the execution of the system's core functions, such as text analysis, processing user requests, and data management. This module uses TextAnalyzer.Core resources to work with base models and provides connectivity to external APIs and adapters.

The TextAnalyzer.Adapters.Driving.Api module is used for the system's interaction with external clients. It is responsible for providing an API through which external applications and services can access text analysis functionality. This module receives data from TextAnalyzer.Application and provides it in a convenient format for external users, ensuring integration with other systems.

The Support Vector Classification (SVC) method was chosen for sentiment analysis due to its high accuracy when working with various datasets and its ability to effectively classify text data. SVC is well-suited for tasks with a small number of classes, as this algorithm creates optimal hyperplanes that separate classes, allowing for better results even with a small amount of training data [8, 9]. Another important advantage of SVC is its robustness to noise in the data.

In the support vector method, the main idea is to find a hyperplane that separates two classes in such a way as to maximize the margin between them. For linearly separable data, a hyperplane is sought that has the equation where w is the weight vector, x is the feature vector, and b is the bias:

$$w^T x + b = 0. \quad (1)$$

The algorithm searches for a hyperplane that maximizes the gap between the points of both classes. This is achieved by minimizing the weight vector w, which is equivalent to maximizing the margin between the classes. The boundary conditions for each point x_i in class y_i (where $y_i = \pm 1$) are defined as:

$$y_i(w^T x_i + b) \geq 1. \quad (2)$$

This means that all points of one class must lie on one side of the hyperplane, and points of the other class on the opposite side, with the nearest

points being at least 1 unit away from the hyperplane. The objective is to minimize the function:

$$\frac{1}{2}\left|\left|w\right|\right|^2. \quad (3)$$

However, if the data is not linearly separable, additional error variables ξ_i are introduced to allow certain points to be misclassified. Then the conditions are modified to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i. \quad (4)$$

And the new function to be minimized:

$$\frac{1}{2}\left|\left|w\right|\right|^2 + C\sum_i \ \xi_i, \quad (5)$$

where C is a hyperparameter that controls the trade-off between maximizing the margin and the number of misclassified points.

When training the machine learning model, the SVC implementation from the sklearn library was used, along with auxiliary libraries such as pandas and nltk [10, 11, 12].

The dataset was loaded from a CSV file, and for simplicity, only two columns were selected from it: text and labels. The labels were converted from numerical to text labels, and then they were encoded into a numerical format using LabelEncoder, which is required for training the model.

To obtain more accurate results, it is important to preprocess the data; for this, English stop words are loaded from the nltk library, and SnowballStemmer is used for stemming [12]. A function was also created for preprocessing text strings:

The data was split into training and test sets using `train_test_split`, which helps avoid overfitting the model, ensures stable results during testing, and allows for validation on a separate dataset. A Pipeline was used to build the model, combining two stages: text vectorization using TfidfVectorizer and training the SVC model with a linear kernel [13, 14]. After training the model, predictions are

made for the test set, and then accuracy is calculated based on the predicted and actual values.

```
    X_train, X_test, y_train, y_test = train_test_split(
        pd.concat([df['text'][:10000],   df['text'][-10000:]]),
pd.concat([df['label'][:10000],          df['label'][-10000:]]),
test_size=0.2, random_state=21910825
    )
    pipeline = Pipeline([
        ('vectorizer',       TfidfVectorizer(max_features=5000,
ngram_range=(1, 2))),
        ('svm', SVC(kernel='linear', probability=True))
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Model Accuracy: {accuracy * 100:.2f}%')
```

The Latent Dirichlet Allocation (LDA) method was chosen for topic modeling due to its ability to automatically identify hidden topics in texts, making it ideal for working with text corpora of various sizes [2]. LDA uses a probabilistic model that identifies a set of topics that best describe the text corpus, allowing for the identification of important topics and their keywords. Thanks to this model, the system can perform in-depth analysis of text data, which is useful for studying structured patterns in large data sets.

The main idea behind LDA is that documents can be represented as mixtures of latent topics, and each topic is represented by a set of words with specific probabilities. LDA searches for these latent topics, which can be used to understand the structure of the text.

Mathematically, for each document *d* and topic *k*, LDA models the probability that a given word belongs to that topic. In this context, a document is represented as a mixture of topics, and each topic is a mixture of words.

The process includes the following steps:

– for each document d, a topic distributionθ_d is selected, which is a Dirichlet parameter;

– for each word in the document, a topic z is selected based on the topic distributionθ_d ;

– a word is selected for topic z based on another Dirichlet distributionϕ_k , which models the probability of words for each topic.

The key formula in LDA is the calculation of a word's probability in a document as the sum across all possible topics:

$$P(W_i|d) = \sum_{k=1}^{K} P(w_i|z_k)P(z_k|d). \quad (6)$$

In the implementation of LDA model training, the main steps include loading data, preprocessing it, vectorizing it, and building the LDA model for topic detection. First, the necessary libraries are imported, including nltk for text processing and sklearn, which provides the LDA model implementation. As with sentiment analysis, the data is read from a CSV file, and for simplicity, only the text column is used.

```
nltk.download('stopwords')

stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = text.split()
    tokens = [stemmer.stem(token) for token in tokens if
token not in stop_words]
    return ' '.join(tokens)
df = pd.read_csv('./data/data.csv', names=['label', 'id',
'date', 'flag', 'user', 'text'])
df = df[['text']]
df['text'] = df['text'].apply(preprocess_text)
```

Once the data is processed, it is split into training and test sets using `train_test_split`, and a Pipeline is created that combines text vectorization using `TfidfVectorizer` with LDA model training via `LatentDirichletAllocation` [13, 14, 15]. Setting the `n_components` parameter to 3 means that the model attempts to identify three main topics in the text.

```
train, test = train_test_split(
    pd.concat([df['text'][:10000], df['text'][-10000:]]),
test_size=0.2, random_state=21910825
)
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(max_features=5000,
ngram_range=(1, 2))),
    ('lda', LatentDirichletAllocation(n_components=3,
random_state=21910825))
])
pipeline.fit(train)
feature_names =
pipeline.named_steps['vectorizer'].get_feature_names_out()
model = pipeline.named_steps['lda']
for topic_idx, topic in enumerate(model.components_):
        top_features_ind = topic.argsort()[:-10 - 1:-1]
        top_features = [feature_names[i] for i in
top_features_ind]
        print(f"Topic {topic_idx}: {top_features}")
```

After training the model, keywords are output for each topic, allowing us to understand the main content of the topics identified by LDA. Using `model.components_`, we can obtain the indices of the most significant words for each topic, which aids in further text analysis. Thus, the training and analysis of the topics provided in the text data are complete, which is an important step in the development of a text analysis system.

The system's core API is implemented in C#, ensuring integration with all necessary functional modules. However, a separate application based on FastAPI has been developed to utilize machine learning models [16; 17]. This approach

allows for more efficient management of specific tasks related to data processing and running machine learning models. The separation of the analytical module into an independent service also corresponds to the principles of microservice-oriented architecture, where individual services communicate through lightweight interfaces and can be maintained or scaled separately [18].

Conclusions. As a result of this study, a conceptual and software framework for a text sentiment analysis system based on thematic modeling has been developed, combining machine learning tools with a web-oriented application architecture. Unlike narrow experimental solutions, the proposed system is not limited to the isolated application of a classification model but covers the entire text processing workflow: data input, preprocessing, analytical processing, secure user interaction, storage of results, and their presentation in the interface. This allows the developed software tool to be viewed as a comprehensive platform for working with text corpora in tasks such as public sentiment monitoring, marketing analysis, and research observations.

Prospective areas for further research include expanding language support, adapting models to domain-specific corpora, improving the quality of thematic modeling, expanding reporting and visualization capabilities, and conducting more detailed experimental evaluation of models on test datasets. Separately, it is worth developing a multi-class sentiment classification scheme, integrating additional mechanisms for explaining results, and exploring hybrid approaches in which classical models are supplemented with modern contextual representations. This direction of development will allow us to maintain the system’s engineering manageability while simultaneously increasing the analytical depth of text data processing.

References

1) Hankar M., Kasri M., Beni-Hssane A. (2025). A comprehensive overview of topic modeling: Techniques, applications and challenges. Neurocomputing. Vol. 628. Article 129638. DOI: 10.1016/j.neucom.2025.129638.

2) Blei D. M., Ng A. Y., Jordan M. I. Latent Dirichlet Allocation. (2003). Journal of Machine Learning Research. Vol. 3. P. 993-1022.

3) Lee D. D., Seung H. S. (1999). Learning the parts of objects by non-negative matrix factorization. Nature. Vol. 401. P. 788-791. DOI: 10.1038/44565.

4) Deerwester S., Dumais S. T., Furnas G. W., Landauer T. K., Harshman R. (1990). Indexing by latent semantic analysis. Journal of the American Society for Information Science. Vol. 41, No. 6. P. 391-407. DOI: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

5) Hayat F., Shatnawi S., Haig E. (2024). Comparative Analysis of Topic Modelling Approaches on Student Feedback. Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR. P. 226-233. DOI: 10.5220/0012890400003838.

6) Devlin J., Chang M.-W., Lee K., Toutanova K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT 2019. Minneapolis. P. 4171-4186. DOI: 10.18653/v1/N19-1423.

7) Grootendorst M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv preprint arXiv:2203.05794. DOI: 10.48550/arXiv.2203.05794.

8) Cortes C., Vapnik V. (1995). Support-vector networks. Machine Learning. Vol. 20. P. 273-297. DOI: 10.1007/BF00994018.

9) Diri G., Obiorah P., Du H. (2025). Comparative Study of Sentiment Analysis Techniques: Traditional Machine Learning vs. Deep Learning Approaches. InSITE 2025. P. 23. DOI: 10.28945/5490.

10) Pedregosa F., Varoquaux G., Gramfort A. et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. Vol. 12. P. 2825-2830.

11) Scikit-learn developers. sklearn.svm.SVC. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (accessed: 21.04.2026).

12) Bird S., Klein E., Loper E. (2009). Natural Language Processing with Python. O'Reilly Media. 504 p.

13) Scikit-learn developers. sklearn.feature_extraction.text.TfidfVectorizer. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (accessed: 07.05.2026).

14) Scikit-learn developers. sklearn.pipeline.Pipeline. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html (accessed: 21.04.2026).

15) Scikit-learn developers. sklearn.decomposition.LatentDirichletAllocation. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html (accessed: 21.04.2026).

16) Microsoft. Create web APIs with ASP.NET Core. Microsoft Learn documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-10.0 (accessed: 22.04.2026).

17) FastAPI. FastAPI documentation. URL: https://fastapi.tiangolo.com/ (accessed: 22.04.2026).

18) Lewis J., Fowler M. (2014). Microservices: a definition of this new architectural term. URL: https://martinfowler.com/articles/microservices.html (accessed: 22.04.2026).