UDC 004.896

HIERARCHICAL CLASSIFICATION OF UKRAINIAN TECHNICAL TEXTS USING TREE-BASED MODELS: APPLICATIONS IN THE AUTOMOTIVE INDUSTRY

S. Mashtalir ORCID: 0000-0002-0917-6622
Kharkiv National University of Radio Electronics
E-mail: sergii.mashtalir@nure.ua
O. Nikolenko ORCID: 0000-0002-6422-7824
Uzhhorod National University
E-mail: oleksandr.nikolenko@uzhnu.edu.ua

Abstract. *Hierarchical classifiers play a crucial role in addressing complex classification tasks by breaking them down into smaller, more manageable sub-tasks. This paper focuses on the technical Ukrainian texts hierarchical classification, specifically the classification of repair works and spare parts used in automobile maintenance and servicing. We tackle the challenges posed by multilingual data inputs – specifically Ukrainian, Russian, and their hybrid – and the lack of standard data cleaning models for the Ukrainian language. We developed a novel classification algorithm, which employs TF-IDF vectorization with unigrams and bigrams, keyword selection, and cosine similarity for classification. Also this paper describes a method for training and evaluating a hierarchical classification model using parameter tuning for each node in a tree structure. The training process involves initializing weights for tokens in the class tree nodes and input strings, followed by iterative parameter tuning to optimize classification accuracy. Initial weights are assigned based on predefined rules, and the iterative process adjusts these weights to achieve optimal performance. The paper also addresses the challenge of interpreting multiple confidence scores from the classification process, proposing a machine learning approach to calculate a unified confidence score. This score helps assess the classification reliability, particularly for unlabeled data, by transforming input values, generating polynomial parameters, and using logarithmic transformations and scaling. The classifier is fine-tuned using hyperparameter optimization techniques, and the final model provides a robust confidence score for classification tasks, enabling the verification and classification results optimization across large datasets. Our experimental results demonstrate significant improvements in classification performance. Overall classification accuracy nearly doubled after training, reaching 92.38%. This research not only advances the theoretical framework of hierarchical classifiers but also provides practical solutions for processing large-scale, unlabeled datasets in the automotive industry. The developed methodology can enhance various applications, including automated customer support systems, predictive maintenance, and decision-making processes for stakeholders like insurance companies and service centers. Future work will extend this approach to more complex tasks, such as extracting and classifying information from extensive text sources like telephone call transcriptions.*

Keywords: *NLP, tree-based classification, machine learning, data analysis, applied intelligent systems.*

1. Introduction and literature review

The field of tree-based data classification has gained increasing prominence due to its capacity to efficiently manage and categorize data with inherent hierarchical structures. This capability is essential for applications across diverse disciplines, such as phylogenetic tree analysis in biology [1], file system organization in computer science [2], and document categorization in information retrieval [3]. The primary strength of tree-based classification lies in its ability to structure data hierarchically, which facilitates efficient retrieval, analysis, and interpretation.

In tree-based classification models, each node within a tree represents a category or class, with branches to child nodes denoting subcategories. The classification process involves assigning data points to appropriate nodes based on their attributes, following a set of predefined rules or models that guide their placement. While various methodologies – such as Decision Trees [4], Random Forests [5], Gradient Boosted Trees [6], and Hierarchical Clustering [7] – have been extensively developed and optimized for numerical and formal categorical attributes, a significant gap exists in their application to Natural Language Processing (NLP).

Although some progress has been made in applying tree-based classification to NLP tasks like fake news detection [8], document clustering using summarization [9], and classification using Discriminative-Semantic Features [10], there is still a lack of robust frameworks for categorizing textual data into predefined hierarchical classes, particularly for less-resourced languages and specialized technical terminologies. Text mining remains a critical area of research due to its potential to produce concise textual representations of complex physical or technical phenomena. As scientific fields evolve and more sophisticated deep learning tools become available, the need for effective big data text analysis has expanded to various applications, such as medical diagnostics [11], where labeling algorithms can parse large-scale textual datasets, and information extraction in noisy environments [12].

Despite the wealth of data available, NLP systems still grapple with issues related to data incompleteness and errors [13, 14]. Inaccurate or incomplete training data can lead to biased or erroneous model outputs, affecting the reliability of NLP applications. Addressing these issues requires the development of more robust data curation and cleaning processes. However, many existing approaches do not account for the unique characteristics of specific languages, relying instead on generic toolkits that limit their efficacy in multilingual or domain-specific contexts.

Consequently, research into NLP methods that incorporate language-specific features has gained considerable interest. Such methods have not only improved human-computer interactions but also broadened the applicability of NLP in various domains. Despite recent advancements, challenges persist, particularly in processing specialized scientific and technical texts that employ complex, domain-specific terminologies. Developing NLP models that can accurately interpret and generate content in these contexts demands further research, including domain-specific fine-tuning and the construction of specialized lexicons. Furthermore, NLP models and libraries are disproportionately developed for major languages, leaving many languages underrepresented and limiting the accessibility of these technologies [15, 16]. The problem is further compounded in multilingual settings, where shared linguistic roots and overlapping vocabulary complicate accurate language processing [17]. Developing robust multilingual NLP models remains active research area.

This study consolidates our previous research on addressing a complex Natural Language Processing (NLP) challenge, focusing on the classification of automotive repair parts and labor descriptions [18-20]. The data utilized for this research was derived from a Garage Management System (GMS).. The dataset is notable for its high variability, containing errors, technical abbreviations, and domain-specific jargon, with entries predominantly in Ukrainian and Russian, and occasionally a blend of the two, known as "Surzhik." Additionally, annotated data is available in the form of a hierarchical classifier for automotive service jobs in both languages.

The primary objective is to categorize each labor entity into predefined categories. To achieve this, we establish a comprehensive data cleansing and preprocessing pipeline, including tokenization (see Figure 1 and [21]), to convert raw text into a structured format suitable for classification. An essential step in this process is vocabulary normalization, which is crucial for managing computational resources and processing time [17]. This paper investigates tree-based classification methodologies for textual data, exploring theoretical foundations, algorithms, and applications, with a particular emphasis on classifying automotive service descriptions. Such methodologies are invaluable for automating processes within the automotive industry, ranging from quality monitoring in service centers to centralizing vehicle issue data for stakeholders like insurance companies and manufacturers.

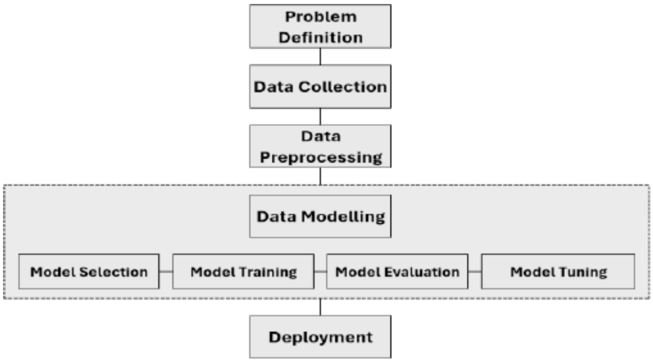

Figure 1. *Typical data science process*

The classification of automotive labor entities falls under the broader category of unstructured data processing, a challenging area that requires a blend of natural language processing expertise and advanced data science techniques [22]. Given the hierarchical nature of the task directory, this problem is framed as a supervised machine learning task, facilitating the development and evaluation of classification models. As illustrated in Figure 1, typical data science workflows are divided into distinct phases: Problem Definition, Data Collection, Data Preprocessing, Data Modeling (encompassing Model Selection, Model Training, Model Evaluation, and

Model Tuning), and Deployment. These phases will be elaborated upon in subsequent sections of the paper.

2. Problem definition and data collection

Automotive sector issues

This work focuses on the technical Ukrainian texts hierarchical classification. The primary issue addressed in the study is the classification of repair works and spare parts used in the automobiles' maintenance and servicing.

This problem's significance cannot be overstated. Out of approximately 1.5 billion vehicles globally, only 20% have detailed, centrally collected, stored, and analyzed repair information [23] and [24]. This data pertains primarily to new vehicles, up to four years old, which are serviced at authorized service centers. Automotive manufacturers utilize specialized software in the original equipment service centers (OES), where all repairs and spare parts are accurately classified. This allows for the collection of precise statistical data on the individual components and assemblies reliability, their operational characteristics, warranty cases, and more [25]. Unfortunately, once a vehicle leaves the official service network, its subsequent repair and maintenance history becomes fragmented and often lost. Non-authorized service centers lack a unified classifier for repairs and spare parts, let alone a single information system. Furthermore, there are thousands of such systems worldwide, each with different languages and data formats. In Ukraine alone, dozens of similar programs are used. Figure 2 illustrates the automotive fleet structure and highlights the data problem concerning repairs.

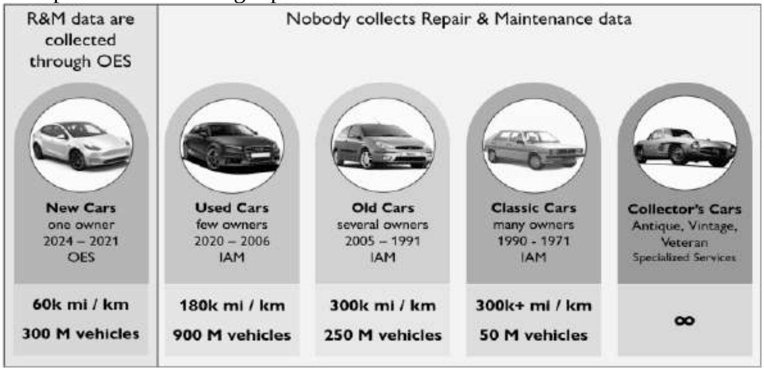

Figure 2. *Global car park structure*

Repair data is valuable not only to automotive manufacturers but also to various other sectors. For example, insurance companies can benefit from this data to determine repair costs and residual vehicle values. Similarly, service centers and even car owners would find it beneficial to have information not only about the current repair costs but also about future expenses related to repair and maintenance.

Data collection and data structure

Our research utilized anonymous data from an online garage management system, covering five years and 500 service stations, yielding over 1 million work records and 1.2 million spare parts records annually. Additionally, customer calls recorded via IP telephony and transcribed into text data contributed to a substantial dataset with hundreds of thousands of texts and tens of millions of data rows. Training and test sets were randomly drawn and manually labeled. The general objective is to develop an algorithm for extracting and further classifying texts using given tree-based classes sets. For this purpose, we have 4 types of datasets:

1. **Classes** – two distinct trees for works (180 entries) and parts (1,600 entries).
2. **Training** – the class trees, augmented with manually labeled data (6,000 entries).
3. **Test** – additional labeled dataset (11,000 entries).
4. **Operational or Input Data** – tens of millions of entries.

Let us examine these types in greater detail.

Car repair works are organized in a three-level tree structure as shown in the data extract in Table 1.

Table 1

Repair works classes data tree extract

ID	Parent ID	UA	EN
1000	0	Діагностичні роботи	Diagnostic work
1100	1000	Діагностика	Diagnostics
1101	1100	Ручна діагностика	Manual diagnostics
1102	1100	Комп'ютерна діагностика	Computer diagnostics
1200	1000	Тестування	Testing
2000	0	Загальні роботи	General works
2100	2000	Заміна	Replacement

Car components are classified within a hierarchical four-level tree structure. The classification begins with the most general component types, such as mechanical and body parts, oils and fluids, wheels and tires. These broad categories are then divided into their corresponding systems, including filters, power transmission, braking, suspension, steering, engine, cooling, electric and electronic systems. Within each system, the classification is further refined into specific components. For example, the suspension is divided into subcategories such as damping, arms, wheel hubs, bearings, etc. Finally, the lowest classification tier consists of specific spare parts detailed lists, such as shock absorbers, struts, coil and leaf springs. The car parts data tree extract is presented in Table 2. The training and test datasets comprise combined repair works and parts lists. For example, an entry might be "Pneumatic damping diagnostics on shock-tester". These lists have been manually labeled specifically for this study. Operational data consists of arbitrary text, which may include information from garage management systems (GMS), phone calls transcriptions, messages from messengers, etc. The objective is to determine whether the input text contains information related to car repairs and to correctly assign it to one of the predefined classes for works and parts.

Table 2

Car parts classes data tree extract

ID	PARENT ID	UA	EN
1000000	0	Механічні деталі	Mechanical parts
1070000	1000000	Амортизація	Suspension damping
1070100	1070000	Амортизатори і стійки	Shock absorbers & struts
1070101	1070100	Амортизаторі підвіски	Shock absorbers
1070102	1070100	Стійки підвіски	Struts
1070105	1070100	Пневмо-амортизатори	Pneumatic shocks
1070300	1070000	Опори амортизаторів	Strut mountings

Classification Success Criteria

A classification is deemed successful if:

1. No less than 90% of car repair works are identified and extracted from the incoming unlabeled texts
2. Of these, no less than 90% of works and parts are correctly assigned to their appropriate classes.

For example, the text "Pneumatic damping diagnostics on shock-tester" should be accurately classified into class 1102 for works and 1070105 for parts

For this study purposes, we simplify the task by assuming that the input text contains information about works and components. Therefore, only the second criterion of successful classification is considered. The task of extracting relevant information about repair works and automotive parts from arbitrary text will be addressed in future studies.

In summary, this work aims to address the critical issue of technical texts hierarchical classification, focusing on the automotive industry. By improving the classification and analysis of repair and maintenance data, we can enhance this information's reliability and accessibility for multiple stakeholders, ultimately contributing to better decision-making and resource management in the automotive sector.

3. Data preprocessing

Classical NLP preprocessing pipeline contains some preliminary (sentence segmentation, word tokenization), frequent (stop word removal, stemming and lemmatization, removing digits/punctuation, lowercasing) and other steps (normalization, language detection, code mixing, transliteration) [17, 21].

In our data preprocessing pipeline, certain modifications were necessitated due to the origin of the data. Specifically, some conventional steps such as sentence segmentation were deemed unnecessary.

Conversely, we introduced additional procedures tailored to the dataset's characteristics, which included language detection, specific Cyrillic characters approach, the disassembling of compounded words and handling of specific shortcuts and abbreviations.

Language Identification

Language identification is accomplished through the utilization of two distinct approaches:

1. **Identification of Specific Characters:** This approach involves the recognition of language-specific characters, such as "і," "ї," "є," and "ґ" for Ukrainian, as well as "ы," "ъ," "э," and "ё" for Russian.
2. **Word Counting in Dictionaries:** A complementary method relies on counting the occurrences of words found in both Ukrainian and Russian dictionaries, as detailed in the "Lemmatization" chapter.

Following the language identification process, all data undergo translation into Ukrainian. This translation is facilitated by two custom correspondence dictionaries, which are elaborated upon in the chapters titled "Translation of Tokens from Russian to Ukrainian" and "Synonyms".

Text Lines Normalization

Text line normalization encompassed a series of standardization procedures. These encompassed segregating numbers and punctuation symbols with spaces, uniformly converting all characters to lowercase, substituting backslashes ("\") with regular forward slashes ("/"), and replacing underscores ("_") with spaces. Additionally, our specific task demanded the normalization of various types of apostrophes into a singular format. This process also extended to the treatment of certain Cyrillic characters, such as transforming "ґ" to "г" and "ё" to "е". Furthermore, prior to the removal of stop words and special characters, common abbreviations featuring slashes or hyphens were substituted with their expanded counterparts, a feature particularly relevant to the context of garage repair texts (e.g., "к-т" denoting "комплект" (kit), "д/м" signifying "демонтаж / монтаж" (mounting / dismounting) and "о/р" representing "охолоджуюча рідина" (cooling fluid) among others). Subsequently, all special characters, with the exception of the apostrophe, which holds linguistic significance in the Ukrainian language, and designated stop words were eliminated from the text.

Stopwords

Our compilation of stopwords comprised a comprehensive set, encompassing both Ukrainian and Russian languages. Notably, certain stopwords present in the general set were excluded, given their relevance to our classification task. For instance, "ТО," an abbreviation for "технічне обслуговування" (technical maintenance), and "ніж," which could be interpreted as a noun (knife) and is pertinent to our directory, were retained. Conversely, additional stopwords were introduced that did not significantly contribute to the subsequent classification process. These included brand names of cars or parts, terms such as "auto," "automobile," "automotive," "service," and other highly generic words devoid of distinctive characteristics relevant to individual entities.

Translation Of Tokens From Russian To Ukrainian

Since we possess detailed classifiers containing Ukrainian and Russian versions of names, and since most names share an identical number and order of words in both versions, we were able to automatically construct a correspondence dictionary between Russian and Ukrainian words.

For each name in the dictionary, we iteratively compare Ukrainian and Russian name versions.

1. Split both versions into token lists.
2. If the lengths of the token lists are equal, iterate through the tokens and increment a "counter" by 1 for each corresponding Ukrainian-Russian token pair. If the number of tokens in the names is not the same, add the missing tokens to the "omitted" category, which is then manually checked. We also skip tokens that are identical in both versions.
3. We obtain correspondence dictionaries where each Russian token corresponds to Ukrainian tokens that were found in the same position in the sentence, along with the number of such occurrences. We select the translation option that occurred most frequently as the most likely correct one. Consequently, we have a dictionary where each Russian token can be matched to its Ukrainian equivalent.
4. Additional steps included manual verification of token translations that differed significantly according to the Jaro-Winkler metric [26], as well as the addition of translations for omitted names with differing token counts.

Tokenization of "Concatenated" Tokens (with Missing Spaces)

The algorithm takes as input a unique set of tokens derived from the data we intend to classify subsequently. It searches for concatenated tokens within this set. For each input token:

1. Check whether it starts or ends with a token known to us.
2. If so, separate it and add it to the "parts" list.
3. Repeat steps 1-2 until we traverse the sorted list of known tokens.
4. Anything that remains unprocessed is added to the list of parts. Remove all parts that are absent from the set of known tokens.
5. If, after this process, more than one part is obtained, identify it as a concatenated token. Add both the original concatenated token and its decomposition to a "dictionary" of token decompositions, which is used later to replace such tokens in input strings.

This token concatenation search is relatively basic in nature, as it can only break tokens that start or end with reference tokens without typographical errors. In the worst-case scenario, a token may not only be concatenated but also contain errors within its constituent parts, in which case the algorithm will fail to identify it. However, concatenated tokens themselves are relatively infrequent, and concatenated tokens with errors are even rarer. Developing a more complex algorithm to address such cases would entail significant computational costs. Therefore, we have chosen to implement this straightforward approach.

Spelling Correction

Spelling errors are identified within tokens that do not exist in reference dictionaries; otherwise, they are considered correct and skipped. Essentially, this process entails the search for the most similar words among those present in the token sets from reference dictionaries, including all their inflected forms found in

Ukrainian and Russian noun and adjective dictionaries. For input, we receive a unique set of tokens from the data that we plan to classify subsequently. Typographical errors are sought within this set. The error detection occurs in two stages.

1. In the first stage, search for the nearest match for tokens that differ by no more than 2 characters from existing tokens. The match is found using the Jaro-Winkler [27] similarity function and is accepted if the similarity value exceeds a predefined threshold. The Jaro-Winkler metric assigns greater weight to token "prefixes" (letters at the beginning of the token).
2. In the second stage, the match is also sought using the same function but without a limit on the maximum of 2 character differences, hence, the threshold value is higher compared to the first stage. Therefore, in the first stage, we tolerate lower "similarity," as long as the token differs from existing ones by no more than 2 characters, while in the second stage, greater dissimilarity is allowed, but the similarity value requirement is higher.

In essence, during the first stage, the distance function allows for smaller "similarity," but tokens must differ from existing ones by no more than 2 characters. In the second stage, greater differences (more than 2 characters) are allowed, but the similarity requirement is stricter.

Motivation For Choosing The Jaro-Winkler Metric

The Jaro-Winkler Measure is a measure of *similarity* / *distance* between two text sequences. It uses the prefix scaling factor *p*, which provides a higher score to sequences that match at the beginning up to a specified prefix length *l*. The higher the Jaro-Winkler similarity measure, the more similar the two text sequences are. The score is normalized such that 0 indicates no similarity, and 1 indicates a perfect match. *Similarity* and *distance* are inversely related, and their correspondence is established by the formula *distance* = 1 - *similarity*. The Jaro-Winkler Measure is a modification of the Jaro measure.

The Jaro similarity sim_j of two text strings s_1 and s_2 is determined by the formula (1):

$$sim_j = \begin{cases} 0 & if\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & if\ m > 0 \end{cases}, \qquad (1)$$

where: $|s_i|$ – length of string s_i

m – number of matching characters

t – number of transpositions.

Two characters from s_1 and s_2 are considered a match if they are identical and located no more than *p* positions apart. If no matches are found, the algorithm stops and returns a *similarity* score of 0. If matches are found, the number of transpositions is then calculated. A transposition occurs when a corresponding (matching) character is not in its correct position, and the number of corresponding characters not in their correct position, divided by 2, yields the number of transpositions.

The Jaro-Winkler similarity sim_{jw} of two text strings s_1 and s_2 is determined by the formula (2):

$$sim_{jw} = sim_j + lp\ (1 - sim_j), \tag{2}$$

where: sim_j – Jaro similarity of text strings s_1 and s_2

l – length of the common prefix at the beginning of the string, maximum of 4 characters

p – a constant scaling coefficient that adjusts the estimate in the direction of increasing values in the presence of a common prefix: the standard value is 0.1.
By substituting formula (1) into the Jaro-Winkler similarity expression, we obtain formula (3):

$$sim_j = \begin{cases} 0 & if\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) + lp\ \left(1 - \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right)\right) & if\ m > 0 \end{cases} \tag{3}$$

There are many distance metrics, among others Levenshtein Distance, Indel (Insertion-Deletion) Distance, Hamming Distance.

Levenshtein Distance. The minimum number of single-character operations (insertions, deletions, and substitutions) required to transform one text string into another [28]:

–	Insertion:	автообіль → авто**м**обіль
–	Deletion:	автом**и**обіль → автомобіль
–	Substituion:	автомо**ь**іль → автомо**б**іль

According to Levenshtein pair of редуктор → редуктор**ний** has a distance of 3 (3 insertions) and pair of ремк**ом**плект → ремк**мо**плект has a distance of 2 (2 substitutions).

Indel (Insertion-Deletion) Distance. The minimum number of insertions and deletions of characters required to transform one text string into another. Substitutions are not allowed, but each substitution can be accomplished by a pair of a deletion and an insertion, making this distance equivalent to the Levenshtein distance with a substitution weight of 2.

Hamming Distance. The number of positions where two strings of equal length differ. It represents the minimum number of substitutions required to transform one string into another and can only be applied to sequences of equal length.

Jaro-Winkler Metric is more complex than simple distance algorithms based on counting basic character operations. It provides a real value between 0 and 1, making the distance values more informative and suitable for comparison and sorting. Additionally, it gives more significance to prefixes, which is a useful property when dealing with "typo searching". Spelling correction is used not for spelling errors only but as well for words with variations in their endings, sharing a common prefix. Giving more weight to prefixes increases the chances of correctly identifying such "typos". Table 3 shows comparative example using the Levenshtein distance and Jaro-Winkler metric, with short and long words. Differences in short sequences are more significant than in long sequences.

Therefore, the metric's value must depend not only on the number of basic operations but also on the length of the sequences, allowing for more accurate word comparisons.

Table 3
Levenshtein distance and Jaro-Winkler Metric Comparison

COMPARABLE STRINGS	LEVENSHTEIN DISTANCE	JARO-WINKLER METRIC
КОЖУХ – КОЖУХА	1	0.966
САЙЛЕНТБЛОК – САЙЛЕНТБЛОКА	1	0.983
КОЖУХ – КОЛЕСА	4	0.577
САЙЛЕНТБЛОК – САЙЛЕТНБЛОКІВ	4	0.951

The Levenshtein metric erroneously yields equivalent distances for the latter two comparisons, which stands in contradiction to the substantial dissimilarity in the lengths of the respective sequences. In stark contrast, the Jaro-Winkler metric aptly delineates the similarities among pairs 1, 2, and 4 while appropriately highlighting the substantial dissimilarity within pair 3.

Lemmatization

Given that we are working with Ukrainian and Russian languages, which have a vast number of word forms, and the unavailability of as sophisticated NLP libraries as for English, we implemented lemmatization independently using electronic dictionaries.

The construction of dictionaries mapping word forms to their lemmas is performed by extracting data from electronic Ukrainian and Russian language dictionaries. During extraction, dictionaries of correspondences for specific word forms to certain lemma are created (for some word forms, multiple lemmas may exist). An inverse dictionary, mapping lemma to word forms, is also generated. Subsequently, for word forms with multiple corresponding lemmas, we choose a single lemma (usually the most frequently occurring one, or in the case of equal occurrences, the first in the list). Word forms corresponding to all other lemmas are attributed to the chosen lemma. Although this approach may result in minor drawbacks when one word form belongs to different parts of speech (e.g., adjectives and nouns in our case), such situations are rare. This approach is preferred to a scenario in which some inflections are lemmatized into one lemma while others into a different one. After completing these steps, we establish a definitive dictionary of correspondences between noun cases and lemmas for use in preprocessing.

Additionally, during lemmatization, another correspondence dictionary mapping Russian lemmas to their Ukrainian counterparts is generated. To achieve this, we traverse the Russian-Ukrainian translation dictionary, searching for lemmas from the form-lemma dictionary described earlier for each pair of words. If lemmas are found for both words, and they are not identical, these lemmas are added to the lemma translation dictionary. Table 5 presents a systematic exposition of the sequential evolution of the initial text as it undergoes the procedures of text cleansing, preprocessing, and lemmatization.

Separation Of Common Prefixes In Compound Words

Prefixes such as "електро" (electro), "пневмо" (pneumo), "авто" (auto), and others, are separated from tokens to standardize different spellings (both combined and separate) of such words. This also enables the linkage of names containing suffixes of compound words written with and without such prefixes (e.g., "пневмонасос" (pneumopump) – which transforms to "пневмо насос" (pneumo pump) - and "насос" (pump) will be treated as similar tokens; otherwise, these two tokens would have been considered entirely different).

Deciphering Abbreviations

We created a file containing common word abbreviations / acronyms and their corresponding expansions. Abbreviations (including those with slashes and hyphens) were manually identified from a large dataset of job names. During preprocessing, tokens representing abbreviations are replaced with their expanded versions.

Synonyms

This is the specific aspect of the Ukrainian language – existence of “Surzhyk” (blending of Ukrainian and Russian words) and the abundance of synonyms.

We compiled a file containing synonyms for words and word combinations. Synonyms were identified during data processing (including during the creation of translation dictionaries from a list of all encountered translations). Some synonyms were also added based on logical considerations.

All synonym words are unified into a single form.

Vocabulary Size Reduction

As previously noted, the final token vocabulary size plays a pivotal role in determining the computational time and machine learning burden incurred in subsequent stages of the pipeline. Consequently, every step we undertake should exhibit a significant reduction in the volume of tokens.

Table 4 illustrates the outcomes of the algorithm when applied to a dataset comprising 10,288 initial sentences. When commencing with an initial count of 6,062 unique tokens, a sequence of preprocessing steps – including normalization, spell-checking, token disassembly, removal of stop-words, translation, lemmatization, and more – results in a reduction to 2,484 tokens, signifying a remarkable 59% decrease in the original vocabulary size. It is worth noting that since computational time exhibits exponential growth in relation to vocabulary size, this 59% reduction in vocabulary size translates to an impressive 84% reduction in both computational time and computational load.

Table 4
Step-by-step Vocabulary Size Reduction

STEP	STAGE	NR. OF TOKENS
1	Initial number of unique tokens	6 062
2	After normalization	4 847
3	Before lemmatization (after normalization, spell-checking, disassembling con-catenated tokens, stop-words, translation)	3 991
4	After lemmatization	2 484

4. Model selection

The three simplest yet most popular algorithms for text data classification are Naive Bayes, k-Nearest Neighbors (kNN), and Logistic Regression [21]. Each of these algorithms possesses a unique set of strengths and weaknesses, especially when applied to Natural Language Processing (NLP) tasks.

Naive Bayes

Naive Bayes is an algorithm based on Bayes' theorem with the assumption of independence among input features. The probability that a text with given features belongs to specific class is calculated using the Bayes' formula [29].

The algorithm advantages include simplicity, efficiency, and the ability to compute multiple classes simultaneously. It is straightforward to understand and implement, performs reliably even with small training datasets (if the independence assumption holds), and is suitable for large datasets. However, the independence assumption often fails in real-world NLP tasks where words are contextually dependent.

k-Nearest Neighbors (kNN)

The kNN is one of the simplest yet effective classification algorithms. Its essence lies in calculating the distance from a given object to all others, with the object being classified as belonging to the most frequent class among its k neighbors. This method advantages include that it requires no training, making kNN a lazy learning algorithm. It can handle nonlinear data and easily adapts to various types of NLP tasks. However, kNN fully exhibits the curse of dimensionality in high-dimensional spaces, where distances between points become less meaningful. Moreover, its speed drastically decreases as the dataset size grows, and the algorithm performs poorly with noisy data.

Logistic Regression

This classification algorithm is based on the logistic function [30]. The logistic or sigmoid function, which is S-shaped, transforms the linear combination of input parameters into a probability in the interval from 0 to 1.

Unlike classical regression, logistic regression is used not for prediction tasks but for classification and probability estimation. In addition to calculating probabilities that can be useful in estimating the classification results confidence, logistic regression advantages include easy scaling to large datasets and the possibility of regularization to avoid overfitting. On the other hand, logistic regression operates under the assumption of a linear relationship between input parameters, which mainly is far from true in NLP tasks. Moreover, it does not determine the importance of features as effectively as tree-based models. In the practical resolution of our task, the k-Nearest Neighbors (kNN) algorithm was selected for the works classification. Regrettably, none of the implemented algorithms demonstrated adequate accuracy in classifying parts. The highest recorded accuracy was 81.55%, achieved by the kNN algorithm. This outcome necessitated the development, training, and evaluation of a bespoke algorithm tailored to the tree-like architecture of the input data.

5. Classification

Works Classification

Considering the fact that the works directory is relatively small, with each record consisting of 1-2 words (during preprocessing 180 initial rows / 358 initial tokens are transformed into 166 tokens), a slightly modified k-Nearest Neighbors (kNN) algorithm demonstrates good results. This algorithm does not have a training phase, so it belongs to lazy algorithms. Instead of training, works directory initialization is provided, which becomes a set of classes. Initialization includes two steps:

1. Synonyms are added to each class name based on input data processing (during review, markup, classification error analysis).
2. Key words are found for each name (initial and synonyms). Key words are those with minimal document frequency (DF) values in the corpus of all class names. In other words, these are words that occur least frequently among all others (usually only once).

The modified algorithm consists of the following steps:

– During classification, a list of preprocessed text strings of work names from the online GMS is input.

– All text strings are transformed by a vectorizer into binary vectors set (1 - token present in the string, 0 - token absent).

– The classifier finds the distances between the input names vectors and the nearest vectors of work class names using cosine similarity metric.

– For each of the nearest work classes, an additional check is made to see if there is a match by keyword; if not, the option is discarded.

– Among the classes with the minimum distance, the one whose token is closest to the beginning of the input name is selected.

Cosine similarity was chosen because it is a widely used similarity measure for real-valued vectors, which is especially important for parts classification. Additionally, cosine has the nice property that it is 1.0 for identical and 0.0 for orthogonal vectors [31]. The cosine similarity [32] between vectors A and B is calculated by the formula (4):

$$S_C(A,B) = \cos(\theta) = \frac{AB}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4}$$

At the algorithm end, tokens corresponding to the selected master-work class name are removed from the input name to leave only tokens corresponding to the spare part name. The method returns a list of pairs ID – work name corresponding to the input names list, as well as a list of cleaned input names for further parts classification.

Parts Classes And Training Sets Vectorization

The parts catalog is significantly more complex than the master-works directory and encompasses approximately 1,600 entries across all levels. Unlike in works, where we did not engage with a tree-like directory structure, the tree plays a crucial role in the parts classification.

In preparation for training and classification, a class vectors tree is constructed. The foundation for this is the parts directory, wherein each tree vertex undergoes the following stages.

Child nodes initialization, "full names" construction

For each node, a set of names is constructed, which includes both the parent's and all the child names – the so-called "full name" (5):

$$\begin{aligned}&full\_name\ = \\ &[node's\ own\ name] + [own\ name\ + \\ &full\_name\ of\ every\ child\ node]\end{aligned} \tag{5}$$

Classes tree vectorization

The classes tree nodes vectorization occurs through the TF-IDF method use, widely employed in text classification. TF (Term Frequency) represents the ratio of the number of occurrences of a chosen word to the total word count of the document, indicating the word's importance within the document. IDF (Inverse Document Frequency) inversely quantifies the frequency with which a word appears across a collection of documents. Utilizing IDF diminishes the weight of commonly used words. The TF-IDF metric [17] is calculated using the formula (6-8):

$$TF - IDF = TF \times IDF \tag{6}$$

$$TF\,(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}, \tag{7}$$

where: $f_{t,d}$ – raw count of a term t in document d

$\sum_{t' \in d} f_{t',d}$ – number of words in document d

$$IDF(t, D) = log \frac{N}{|\{d \in D : t \in d\}|}, \tag{8}$$

where: N – total number of documents in the corpus, $N = |D|$

$|\{d \in D : t \in d\}|$ – number of documents, where term t appears, i.e. $TF\,(t, d) \neq 0$

For each node in the class tree, a unique matrix is constructed. The matrix rows represent the corresponding TF-IDF values of each token from the child nodes of the given node. Herein, the text for TF calculation is the full name of the parent node, while the corpus for IDF calculation consists of the child nodes full names set. Tokens are considered to be unigrams, direct and inverted bigrams – as the simplest options for the augmented dataset.

Selecting keywords and super keywords

Subsequent steps involve selecting keywords and super keywords. During the training, these receive additional weight, significantly aiding in classification accuracy.

Keywords are identified based on the document frequency values of tokens in each of the child names. Keywords are those with a DF value of 1, meaning they occur in only one of the neighboring child classes.

If a node's name consists of only two words, it automatically becomes a super keyword, receiving even greater weight.

Training data vectorization

At this stage, for each input training string, a matrix is created using the same algorithm as for the class vectorizer.

The matrix rows are the corresponding TF-IDF values of the input string on the specific class node text corpus.

The augmented dataset set is created by tokenizing not only unigrams and bigrams but also bigram permutations. Such input string tokens bigram enrich the feature sets, as different word orders can have the same meaning but be found in different classes. For example, from "brake pads and discs," bigram permutations like "brake pads," "brake discs," "pads brake," "discs brake," "pads discs," "discs pads" are formed.

Now, everything is prepared for initiating the training process.

Parts classification algorithm

Prior to commencing the model training, let us examine the classification algorithm itself. If the model construction occurred in a bottom-up fashion, with each parent node encompassing information about all its children, the classification process unfolds in a top-down manner. Initially, an appropriate class at the tree first level is selected, followed by selection among child nodes at the second level, and so forth. The classification foundation rests on finding the node with the nearest vector. The vectors compared are the weighted rows of the class node matrix and the corresponding input string matrix row. It is reminded that the rows of the class node matrix correspond to the weighted TF-IDF values of tokens from child nodes. Hence, the nearest row in the matrix corresponds to a specific child node, which will determine the next step in the classification. Similar to the works classification, the distance between vectors is determined using cosine similarity. This metric's capability to work with real numbers becomes particularly useful here. Moreover, the value calculated through cosine similarity can be interpreted as the probability of classifying the input string into that specific class, enabling a confidence score computation. The confidence score is calculated as the ratio between the most likely class probability, chosen as the classification result, and that of the second most probable class. If the second class probability is 0, meaning no matches were found in any of the child classes and they are all zeros, or a match was found only in one class and all others are zeros, then a confidence is assigned an arbitrarily high value, for example, 1000.

6. Model training

Our prior chapter delineated the comprehensive process of hierarchical classification for automotive works and parts. However, constraints on space precluded detailed discussions of the training and evaluation processes, despite their critical importance in the classification algorithm, which significantly enhanced the classification accuracy.

Additionally, the confidence scores computation and optimization were not thoroughly examined. Given that 99% of our data was unlabeled, the confidence issue was especially pivotal in our study.

This paper addresses these omissions by providing a detailed training and scoring processes exposition in the subsequent two chapters.

Following the tree construction and the initialization of the vectorizer and classifier for each node, a parameter tuning method for the tree is initiated, which in turn launches, in several threads, the parameter tuning method within each node, acting as potential sub-classifiers of our tree.

Prior to the parameter tuning iterations commencement, several preparatory steps are undertaken:

1. A training dataset is initialized, upon which each parameter values set will be evaluated at each iteration. This set includes input data – vectorized names from the sets comprising the full name of child nodes, as well as additional manually annotated names (which are given greater weight), and the output data – the corresponding child nodes and annotations classes.
2. An initial parameter values set at the node is evaluated. The evaluation function launches a one-step training data classification on the node classifier and checks the accuracy percentage of the resulting classes against the true class values.

The training occurs through the weights (parameters) optimization for the node's matrix tokens in the class tree, as well as weights (parameters) for the input (training) vector.

Initially, all class node matrix elements are assigned preliminary weights according to the following rules:

– Super keyword – 5.0, keyword – 2.5
– Direct child tokens – 2.0 (direct descendants tokens are given more attention than those of further descendants)
– Bigrams – 1.5,
– The first token in the name – 1.5
– Adjectives – 0.5
– Others – 1.0

The exact values for parameters during initialization are not critically important. What matters is that they are greater than 1 or less than 1, and subsequently, the iterative training algorithm will determine the optimal weights.

As with the matrices for class tree nodes, initial weights are determined for the input strings matrices. However, different rules apply here:

– Bigrams – 1.5
– The first token in the name – 1.5
– Tokens created from words in parentheses – 0.5
– Bigrams created from words on the edge of parentheses from both sides – 0.0

Parameter tuning iterative cycle then commences. If it does not conclude within 20 iterations, it halts at the last result. At each iteration, a parameter tuning step is performed:

1. For each parameter, its values are iterated from a possible values predefined set (for example, for most weights >1, parameters from 1 to 10 are iterated in steps of +0.5, and for weights < 1, parameters from 1 to 0 in steps of -0.05).
2. As we are changing parameter values, for each of the training rows, a re-initialization of the weighting parameters is pre-launched, as well as a re-vectorization of the input names (if the value of a parameter related to input vectorization was changed) or a re-initialization of the classifier (if parameters based on which the classifier vectors are built were changed).
3. These values are then evaluated on the training data – through classification and calculating Accuracy – the matches percentage between found and real classes.
4. The parameter and its value that achieve the maximum classification accuracy rating are selected.
5. A check for value update is performed.

– If a change in parameter value led to an increase in accuracy compared to the previous iteration, or accuracy remained the same but the parameter value became closer to 1 – update the node parameter values and proceed to the next iteration;

– If the parameter values iteration did not find a better value for any of the parameters – stop the cycle.

After completing the parameter tuning method on all nodes, the tree can be considered "trained" and used for further classification.

7. Unified confidence score for labeled and unlabeled data

Following classification, a pertinent question remains: how confident are we in its correctness? This is particularly relevant for unlabeled data, as well as for automated decision-making systems.

As noted, the classification result provides us with a classes set along with their probabilities, and confidence scores for each node of the tree. The challenge arises in how to accurately interpret multiple confidence scores. Simple dimensionality reduction methods, such as arithmetic mean or root mean square, which might intuitively be considered, lose crucial information from the tree structure.

In other words, is it better to have confidence closer to the tree's roots or its leaves? Which set provides greater overall confidence, (1000, 0.1, 0.1) or (0.1, 0.1, 1000)? If we were classifying city names, moving through the tree from country to state/region to city, then the set (1000, 0.1, 0.1) would imply high confidence in the country but not in the specific city, whereas (0.1, 0.1, 1000) indicates that we correctly identified Odesa, but it's unclear where exactly – in Ukraine or Texas.

To address this issue, labeled data from the training set are used, on which traditional machine learning is conducted based on three confidence parameters using standard algorithms from the powerful Python library Scikit[33].

In our case, the machine learning process consisted of the following stages.

Parameters Engineering

In many machine learning algorithms, transforming input values is a necessary condition without which the algorithm won't converge to an optimal result due to excessively extreme input values or too significant difference between feature magnitudes. Moreover, most machine learning models train better and faster on as standardized data as possible.

– Clipping is performed (values less than a set minimum become the minimum, and those greater than a set maximum become the maximum) within a range from min=0.000001 to max=1000, to eliminate zero values and the most significant outliers over 1000.

– To capture not only linear dependencies between input parameters and the predicted value but also potential nonlinear input data behaviors, as well as to account for interactions between different input parameters, polynomial parameters up to degree 3 are generated. For example, from input parameters x_1, x_2, x_3, polynomial parameters x_1, x_1^2, x_1^3, x_1x_2, x_1x_3, $x_1^2x_2$, $x_1^2x_3$, $x_1x_2x_3$, x_2, x_2^2, $x_2^2x_1$, etc., are formed.

– Logarithmic transformation of polynomial parameters is conducted to reduce the distribution positive skewness, where most values are relatively close to 0, but some highest values reach up to 109, thereby having a long "tail" to the right, and to bring them to values closer to each other and closer to 0.

– Values are standardized using a scaler. Typically, values are scaled relative to the mean and variance. In our case, RobustScaler from Scikit was chosen as the scaler, which is more resistant to outliers and uses the median and interquartile range instead of the usual mean and variance.

Training

GradientBoostingClassifier [34] was chosen as the classifier, which conducts classification based on boosted trees [35].

In practical applications, effectively deploying the GradientBoostingClassifier necessitates the careful adjustment of its hyperparameters, which play a critical role in shaping the model's accuracy and efficiency. This adjustment process typically involves empirical optimization, where methods such as grid search or random search are frequently employed to identify the most suitable hyperparameter settings.

The hyperparameters selected were:

– n_estimators – the simple models number (decision trees) that make up the ensemble

– learning_rate – the value that indicates how significant the contribution of each model in the ensemble is to the overall result

– max_depth – the maximum depth of the decision trees in the model

– max_features – the maximum number of features considered during the tree nodes splits

– min_samples_split – the minimum number of data points in a node (node samples) required to split a node

– subsample – the fraction of data (among all training data) taken for training each of the simple trees

Hyperparameter tuning was performed using GridSearch, i.e., trying all possible parameters combinations among given values sets with cross-validation.

The hyperparameters quality was assessed using BrierScoreLoss, which shows the average squared difference between the predicted class probability (value pred_proba of the model GradientBoostingClassifier, from 0 to 1, corresponding to how confident the model is that the outcome to which the obtained uncertainty scores correspond is correct) and the true accuracy (0 or 1, depending on the correctness of the classification on training data).

Classification

The GradientBoostingClassifier from the Scikit library is a robust classification algorithm for machine learning tasks, based on the boosting technique. Boosting is an ensemble method that constructs a series of models sequentially, with each subsequent model aiming to correct its predecessors’ errors.

Initially, a decision tree model is created, typically a simple one. This model is imperfect, with accuracy slightly better than a random choice. The first step is not crucial; the iterative process is expected to significantly enhance it.

Next, a loss function is determined to evaluate the model's effectiveness. In this case, the function measures the discrepancy between predicted probabilities and actual class labels, specifically the deviation loss between them.

Gradient boosting methodically enhances the model. At each new step, new models are created to rectify the existing ensemble deficiencies:

– The loss function gradient based on the current model predictions is calculated. This gradient indicates the direction in which predictions should be altered to reduce loss.

– A new decision tree is trained to forecast these gradients for each item in the training set. This tree aims to predict the previous model errors.

– This new decision tree is added to the ensemble with a coefficient known as the learning rate. This coefficient controls the speed at which the model learns. The learning rate is a critically important hyperparameter in gradient boosting. It assesses and scales each tree contribution. If it is too high, the model may overfit; if too low, the model may require too many trees to converge to a satisfactory solution.

– The algorithm continues to add trees until the specified number of trees (n_estimators) is reached or until no further improvement can be made on the training set.

Boosted trees are prone to overfitting. Therefore, several regularization techniques are integrated into the GradientBoostingClassifier:

– Limiting the depth of trees with max_depth

– A fraction of the training data (subsample) is randomly selected to train each tree. This randomness enhances the model robustness.

– Learning rate reduction – the learning_rate parameter scales the contribution of each tree, lowering the overfitting risk by diminishing the updates.

The data obtained after training allow for calculating a single confidence score for unlabeled data. The obtained confidence scores can be sorted from top to bottom. In doing so, homogenous names will have the same score and be located nearby, which is convenient for verifying the classification correctness. If the result is correct/incorrect for one name, it will be the same for all similar names. This allows for creating new classes or optimizing the algorithm immediately for a large number of input data. Table 5 shows the top and bottom five results of the parts classification, confidence scores by tree levels, and the final unified confidence.

Table 5
Top and bottom five parts classification results

TESTING DATA SAMPLE	**LABELED**	**PREDICT.**	**RES.**	**CONFIDENCE BY LEVELS**			
				L-1	**L-2**	**L-3**	**Unif.**
заміна зовн. ручки і приводу замка чи двері	2011300	2011300	True	36	119	97	99.9%
зняття і установка консолі склоочисника	1200500	1200500	True	66	146	73	99.8%
замена сцепного шкворня	1050900	1050900	True	31	670	100	99.8%
установка обігрівального елементу сидіння	2030200	2030200	True	21	1,169	100	99.8%
зняття та встановлення маховика інерційн.	1080300	1080300	True	100	198	100	99.8%
…	…	…	…	…	…	…	…
заміна газонаповнених амортизат. капота	2010300	2011100	False	2	18	3	7.5%
ремонт клапана привода передней двери	2011300	1140400	False	6	5	15	5.5%
ремонт пжд	1160900	1120000	False	156	1	1	5.3%
замена клапана моторного тормоза	1031600	1140400	False	376	4	17	4.4%
проверка клапана моторного тормоза	1031600	1140400	False	376	4	17	4.4%

8. Results achieved and conclusions

Based on proposed approach, a function library in Python was developed. The brief classification times – up to 125 ms for a single row and up to 56 seconds for eleven thousand rows – permit the use of the algorithm in an online mode for wide variety of problems. Accelerations by more than an order of magnitude are achieved for data comprising thousands of rows, thanks to powerful Python algorithms optimized for working with large matrices. The library we developed is also optimized for rapid computation of large data arrays and utilizes all built-in Python optimization techniques. The classification accuracy varied across different datasets from 85% to 98% for works and from 87% to 96% for parts names. As shown in Table 6, the overall classification accuracy of the proposed algorithm nearly doubled after training, reaching 92.38%. The classification of works related to mechanical parts was most effective, while the classification of specialized tasks, such as transmission repair or truck repair works, was less accurate. The partial attribution of this variability to the incomplete directories for certain tasks points towards an potential enhancement area through the expansion and refinement of class directories. One of the significant ancillary benefits observed from our algorithm implementation is the missing terms identification that necessitate inclusion in the directories, thereby improving comprehensiveness and the classification system accuracy. This outcome also contributes valuable insights for domain-specific knowledge bases.

Table 5

Top and bottom five parts classification results

TESTING DATA SAMPLE	**LABELED**	**PREDICT.**	**RES.**	**CONFIDENCE BY LEVELS**			
				L-1	**L-2**	**L-3**	**Unif.**
заміна зовн. ручки і приводу замка чи двері	2011300	2011300	True	36	119	97	99.9%
зняття і установка консолі склоочисника	1200500	1200500	True	66	146	73	99.8%
замена сцепного шкворня	1050900	1050900	True	31	670	100	99.8%
установка обігрівального елементу сидіння	2030200	2030200	True	21	1,169	100	99.8%
зняття та встановлення маховика інерційн.	1080300	1080300	True	100	198	100	99.8%
…	…	…	…	…	…	…	…
заміна газонаповнених амортизат. капота	2010300	2011100	False	2	18	3	7.5%
ремонт клапана привода передней двери	2011300	1140400	False	6	5	15	5.5%
ремонт пжд	1160900	1120000	False	156	1	1	5.3%
замена клапана моторного тормоза	1031600	1140400	False	376	4	17	4.4%
проверка клапана моторного тормоза	1031600	1140400	False	376	4	17	4.4%

The research presented in this paper has successfully demonstrated the application of tree-based classification methodologies to the domain of Ukrainian technical text analysis, specifically focusing on the automotive industry.

Through the development of a Python function library, we have showcased our proposed approach capability to efficiently classify technical texts related to automotive repairs and parts, achieving classification times that support real-time application scenarios. This efficiency opens the algorithm up for a wide array of practical uses, from enhancing the call centers operational quality to the creation of automated chatbots and digital assistants for service advisors in automotive service stations.

In conclusion, the research underscores the profound potential of tree-based classification in navigating the complexities of technical text analysis within the automotive sector. By bridging the gap between structured data classification and the nuanced realm of natural language processing, we pave the way for advanced applications that could significantly impact various stakeholders, including insurance companies, automobile manufacturers, and vehicle owners as shown on Figure 3.

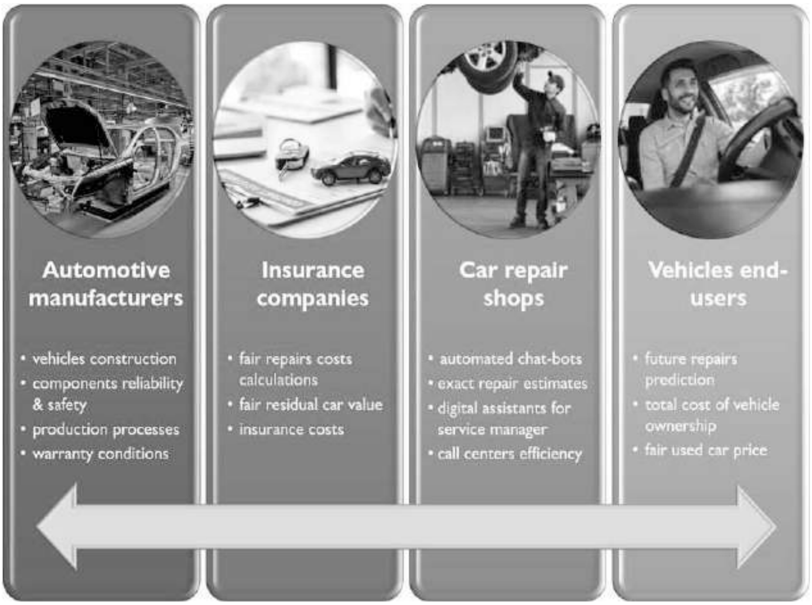

Figure 3. *Practical implementation of automotive works and parts accurate classification*

The ability to accurately predict maintenance costs and reliability of vehicle components from aggregated, labeled big data represents a substantial stride towards demystifying the vehicle ownership total cost, thereby empowering consumers and industry players alike with valuable, actionable insights.

From the perspective of automotive manufacturers, this approach could substantially impact vehicle design, component reliability and safety, production processes, and warranty policies. Insurance companies may benefit from precise repair cost calculations and accurate assessments of residual vehicle value, leading to reduced expenses.

Automotive repair shops can enhance their services by implementing automated chatbots and digital assistants for service managers. Additionally, car owners will be able to determine not only the purchase price of a vehicle but also the total cost of ownership for specific model.

Table 6
Parts names classification results

MODEL TYPE	VECTORIZATION	ACCURACY, TRAINING DATA	ACCURACY, TEST DATA
custom model without weighting and training	count vectors	—	0.5174
	TF-IDF vectors	—	0.6684
main model with weighted parameters after training	Count vectors	0.9365	0.9184
	TF-IDF vectors	0.9552	0.9238

Looking forward, we aim to extend our research to encompass more complex tasks, such as the extraction, identification, and classification of automotive-related works from extensive text bodies, including transcriptions of telephone calls.

9. References

[1] Zhang, Z., Guo, K., Pan, G.: Improvement of phylogenetic method to analyze compositional heterogeneity. BMC Syst Biol 11 (Suppl 4), 79 (2017).
[2] Albadri, N., Dekeyser, S.: A novel file system supporting rich file classification. Computers and Electrical Engineering, vol. 103 (2022)
[3] Kim, SW., Gil, JM.: Research paper classification systems based on TF-IDF and LDA schemes. Hum. Cent. Comput. Inf. Sci. 9, 30 (2019)
[4] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, Inc. (1993)
[5] Breiman, L.: Random Forests. Machine Learning 45, pp. 5–32 (2001)
[6] Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Ann. Statist. 29 (5), pp. 1189 - 1232 (2001)
[7] Aggarwal, C.C., Reddy, C.K.: Data Clustering: Algorithms and Applications. Chapman and Hall/CRC (2013)
[8] Bauskar, S., Badole, V., Jain, P., Chawla, M.: Natural Language Processing based Hybrid Model for Detecting Fake News Using Content-Based Features and Social Features. International Journal of Information Engineering and Electronic Business (IJIEEB), vol.11, No.4, pp. 1-10 (2019).

[9] Gupta, A., Kaur, M., Bajaj, A., Khanna, A.: Entailment and Spectral Clustering based Single and Multiple Document Summarization. International Journal of Intelligent Systems and Applications (IJISA), Vol.11, No.4, pp.39-51 (2019)
[10] Parsafard, P., Veisi, H., Aflaki, N., Mirzaei, S.: Text Classification based on Discriminative-Semantic Features and Variance of Fuzzy Similarity. International Journal of Intelligent Systems and Applications (IJISA), Vol.14, No.2, pp.26-39 (2022)
[11] Trivedi, H., Panahiazar, M., Liang, A., Lituiev, D., Chang, P., Sohn, J., Chen, Y., Franc, B., Joe, B. & Hadley, D. “Large scale semi-automated labeling of routine free-text clinical records for deep learning”. Journal of Digital Imaging. 2018. [Scopus]. DOI: https://doi.org/10.1007/s10278-018-0105-8.
[12] Nguyen, H. & Patrick, J. “Text mining in clinical domain: dealing with noise “KDD '16: Proceedings of the 22nd ACM SIGKDD.” International Conference on Knowledge Discovery and Data Mining. August 2016. p. 549–558. DOI: https://doi.org/10.1145/2939672.2939720.
[13] Arenas, M., Botoeva, E., Kostylev, E. & Ryzhikov, V. “A note on computing certain answers to queries over incomplete databases”. In: CEUR Workshop Proceedings. Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web. Montevideo: Uruguay. 2017.
[14] Li, Y., Currim, F. & Ram, S. “Data completeness and complex semantics in conceptual modeling”. The Need for a Disaggregation Construct. Journal of Data and Information Quality. 2022; 14 (4), Article No. 22: 1–21. [Scopus]. DOI: https://doi.org/10.1145/3532784.
[15] Sebastian, M. P., & G, S. K. “Malayalam natural language processing: challenges in building a phrase-based statistical machine translation system”. ACM Transactions on Asian and Low-Resource Language Information Processing. 2022; 22 (4), Article No. 117: 1–51. [Scopus]. DOI: https://doi.org/10.1145/3579163.
[16] Butnaru, A.-M. “Machine learning applied in natural language processing”. ACM SIGIR Forum . June 2020; 54 (1), Article No. 15: 1–3. DOI: https://doi.org/10.1145/3451964.3451979.
[17] Vajjala, S., Majumder, B., Gupta, A. & Surana, H. “Practical natural language processing: A comprehensive guide to building real-world NLP systems”. Published by O’Reilly Media, Inc. 2020.
[18] Mashtalir, S., Nikolenko, O., “Data preprocessing and tokenization techniques for technical Ukrainian texts”, Applied Aspects of Information Technology. Vol. 6 No. 3 (2023), 318-326. doi: 10.15276/aait.06.2023.22
[19] Mashtalir, S., Nikolenko, O., “Advancing Automotive Technical Text Analysis: A Tree-Based Classification Approach for Ukrainian Texts”, ICCSEEA2024 (2024), in press.
[20] Mashtalir, S., Nikolenko, O., “Optimizing Hierarchical Classifiers with Parameter Tuning and Confidence Scoring”, Applied Aspects of Information Technology, in press.
[21] Jurafsky, D. & Martin, J. “Speech and language processing”. Third Edition draft. 2018. – Available from: https://web.stanford.edu/~jurafsky/slp3/ed3book_jan72023.pdf.

[22] Cielen, D., Meysman, A., Ali, M.: Introducing Data Science: Big Data, Machine Learning, and more, using Python tools. Manning Publications (2016)

[23] Mohammad, A. "Using Blockchain for Data Collection in the Automotive Industry Sector: A Literature Review". Journal of Cybersecurity and Privacy. 2022. 2 (2). DOI: 10.3390/jcp2020014

[24] Danielkiewicz, R. & Dzieńkowski, M. "Analysis of user experience during interaction with automotive repair workshop websites". Journal of Computer Sciences Institute. 2024. Vol. 30: 39-46. DOI: 10.35784/jcsi.5416

[25] Hemphill, T., Longstreet, P. & Banerjee, S. "Automotive repairs, data accessibility, and privacy and security challenges: A stakeholder analysis and proposed policy solutions". Technology in Society. 2022. Vol. 71(3). DOI: 10.1016/j.techsoc.2022.102090

[26] Winkler, W. E. "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage". Proceedings of the Section on Survey Research Methods. American Statistical Association. 1990. p. 354–359. – Available from: https://files.eric.ed.gov/fulltext/ED325505.pdf.

[27] Cohen, W. W., Ravikumar, P. & Fienberg, S. E. "A comparison of string distance metrics for name-matching tasks (PDF)". KDD Workshop on Data Cleaning and Object Consolidation. 2003; 3: 73–78. – Available from: https://www.cs.cmu.edu/afs/cs/Web/People/wcohen/postscript/kdd-2003-match-ws.pdf.

[28] Levenshtein, V. I. "Binary codes capable of correcting deletions, insertions, and reversals". Cybernetics and Control Theory. 1966; 10 (8): 707–710.

[29] Hand, D.J., Yu, K.: Idiot's Bayes – not so stupid after all?. International Statistical Review. Vol 69 part 3, pp. 385 - 399 (2001)

[30] Hosmer, D. W., Lemeshow, S.: Applied Logistic Regression. 2nd edn. John Wiley & Sons, Inc. (2000)

[31] Singhal, A.: Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4), pp. 35–43 (2001)

[32] Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley (2005)

[33] Müller, A. & Guido, S. "Introduction to Machine Learning with Python: A Guide for Data Scientists". 1st ed. Published by O'Reilly Media. 2016.

[34] Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems". 3rd ed. Published by O'Reilly Media. 2022.

[35] Bastos, J. "Predicting Credit Scores with Boosted Decision Trees". Forecasting, 2022. Vol. 4(4): 925-935. DOI: 10.3390/forecast4040050

ІЄРАРХІЧНА КЛАСИФІКАЦІЯ УКРАЇНСЬКИХ ТЕХНІЧНИХ ТЕКСТІВ ЗА ДОПОМОГОЮ ДЕРЕВОВИДНИХ МОДЕЛЕЙ: ЗАСТОСУВАННЯ В АВТОМОБІЛЬНІЙ ПРОМИСЛОВОСТІ

С. Машталір ORCID: 0000-0002-0917-6622
Харківський національний університет радіоелектроніки
E-mail: sergii.mashtalir@nure.ua
О. Ніколенко ORCID: 0000-0002-6422-7824
Ужгородський національний університет
E-mail: oleksandr.nikolenko@uzhnu.edu.ua

Анотація. *Ця стаття присвячена ієрархічній класифікації технічних українських текстів, зокрема класифікації ремонтних робіт і запасних частин, які використовуються для технічного обслуговування та обслуговування автомобілів. Ми вирішуємо проблеми, пов'язані з багатомовним введенням даних Розроблен новий алгоритм класифікації, який використовує векторизацію TF-IDF за допомогою уніграм і біграм, вибір ключових слів і косинусну подібність для класифікації. Описано метод навчання та оцінки моделі ієрархічної класифікації з використанням налаштування параметрів для кожного вузла в структурі дерева. Процес навчання передбачає ініціалізацію вагових коефіцієнтів для токенів у вузлах дерева класів і вхідних рядках з подальшим ітеративним налаштуванням параметрів для оптимізації точності класифікації. Початкові ваги призначаються на основі попередньо визначених правил, і ітераційний процес коригує ці ваги для досягнення оптимальної продуктивності. У документі також розглядається проблема інтерпретації кількох оцінок довіри з процесу класифікації, пропонуючи підхід машинного навчання для обчислення уніфікованої оцінки довіри. Ця оцінка допомагає оцінити надійність класифікації, особливо для немаркованих даних, шляхом перетворення вхідних значень, генерації поліноміальних параметрів і використання логарифмічних перетворень і масштабування. Класифікатор налаштовується за допомогою методів оптимізації гіперпараметрів, а остаточна модель забезпечує надійний показник достовірності для завдань класифікації, уможливлюючи перевірку та оптимізацію результатів класифікації для великих наборів даних. Експериментальні результати демонструють покращення ефективності класифікації. Загальна точність класифікації зросла майже вдвічі після навчання, досягнувши 92,38%. Це дослідження надає практичні рішення для обробки великомасштабних немаркованих наборів даних в автомобільній промисловості. Розроблена методологія може покращити різні додатки, включаючи автоматизовані системи підтримки клієнтів, прогнозне обслуговування та процеси прийняття рішень для зацікавлених сторін, таких як страхові компанії та сервісні центри. Майбутня робота поширить цей підхід на більш складні завдання, такі як вилучення та класифікація інформації з обширних текстових джерел, таких як розшифровка телефонних розмов.*

Ключові слова: *NLP, деревовидна класифікація, машинне навчання, аналіз даних, прикладні інтелектуальні системи.*