



PDF Download
3674912.3674935.pdf
04 February 2026
Total Citations: 0
Total Downloads: 82

Latest updates: <https://dl.acm.org/doi/10.1145/3674912.3674935>

RESEARCH-ARTICLE

Classification of Short Noisy Text

EMANUELA MITREVA, The Bulgarian Academy of Sciences, Sofia, Bulgaria

VLADIMIR GEORGIEV, American University in Bulgaria, Blagoevgrad, Blagoevgrad, Bulgaria

ALEXANDRA NIKOLOVA, The Bulgarian Academy of Sciences, Sofia, Bulgaria

Open Access Support provided by:

The Bulgarian Academy of Sciences

American University in Bulgaria

Published: 14 June 2024

[Citation in BibTeX format](#)

CompSysTech '24: International
Conference on Computer Systems and
Technologies 2024
June 14 - 15, 2024
Ruse, Bulgaria

Classification of Short Noisy Text

Emanuela, EM, Mitreva
Institute of Mathematics and
Informatics, Bulgarian Academy of
Sciences
emitreva@gmail.com

Vladimir, VG, Georgiev
American University in Bulgaria
vgeorgiev@aubg.bg

Alexandra, AN, Nikolova
Institute of Mathematics and
Informatics, Bulgarian Academy of
Sciences
alxnkolova@gmail.com

ABSTRACT

In this paper the problem we are trying to solve is to classify short texts that correspond to task assignments, so that we are able to provide some level of personalization in a framework that provides educational courses. The solution will check what assignments were incorrectly solved by the users and additional ones of the same kind will be offered. However, firstly the texts need to be labeled, so that the system knows which are similar and to offer tasks with the same category. There were several challenges due to the nature of the assignments – short noisy texts with a few records for each category. To overcome them the records were cleaned, lemmatized and encoded before using different classifiers to select the label for each record. Our study investigates the optimal classifier and the parameters tailored to our dataset characteristics.

CCS CONCEPTS

• **Supervised learning by classification;** • **Bayesian analysis;** • **Support Vector Machines;**

KEYWORDS

Short text classification, Support Vector Machine

ACM Reference Format:

Emanuela, EM, Mitreva, Vladimir, VG, Georgiev, and Alexandra, AN, Nikolova. 2024. Classification of Short Noisy Text. In *International Conference on Computer Systems and Technologies 2024 (CompSysTech '24)*, June 14, 15, 2024, Ruse, Bulgaria. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3674912.3674935>

1 INTRODUCTION

In this article we are trying to improve a framework being developed to offer courses to students. The improvement that we need to introduce is to be able to offer additional assignments that are similar to assignments that the students solve incorrectly. For this to be possible the system needs a way to “know” that an assignment is similar to another assignment and not all assignments have categories or labels. One of the ways to do that is through text classification and in our case the classification of short texts that are very noisy – the texts contain not just the assignment, but html tags, etc.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CompSysTech '24, June 14, 15, 2024, Ruse, Bulgaria

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1684-3/24/06

<https://doi.org/10.1145/3674912.3674935>

Text classification is the procedure of assigning predefined labels to text and is an essential and important task in many natural language processing (NLP) applications, topic categorization [1] [2]. There are many approaches to classify text, however when a short text needs to be classified the task is more difficult. Due to the explosive growth of short texts on various social media platforms, classifying short text streams is becoming an increasingly significant problem, but unlike traditional text streams, short text stream data has the following characteristics: short length, weak signal, high volume, high speed, topic shifting, and others [1].

Currently our dataset consists of a little less than 300 records split in 24 different categories, evenly distributed, so a little over 10 records for each category, which raises the question whether classification is even needed, considering that any setup with so few records might be overfitted on those records. But we acknowledge that the accuracy of manual text classification can be easily affected by human factors such as fatigue and experience and we would like to expand the number of records in the system and automate their classification upon inserting in the system. It is desirable to use machine learning methods to automate the text classification procedure to obtain more reliable and less subjective results [3].

2 BASIC ALGORITHM AND PRE-PROCESSING STEPS

The basic algorithm to classify text is:

- Pre-process the text – clean the text, lemmatize it and remove any stop words.
- Format the text in the proper text representation.
- Use a proper algorithm and classifier.
- Assess the accuracy of the classifier that is used.

2.1 Pre-process the text

In general, data pre-processing is not a trivial task - any error or additional data that would prevent the algorithm from running correctly must be cleaned up - sometimes this represents a large portion of the data. Because of the specifics of our data, we had several problems with cleaning the data. The first problem was that the text contains many special characters - html tags and various formatting characters that are added automatically when the data is entered through the system interface. So, the first pre-processing we did was to remove any html tags and other formatting data.

Another problem we had when cleaning the data was which words to be removed, or so-called stop words. Initially, pre-built dictionaries of words were examined [4]. Unfortunately, such a dictionary of words would be useful in larger texts and documents, where the words listed there further increase the overall vocabulary without bringing any additional value and removing them would

make the classifier more effective. In our case, however, due to the specificity of the texts, many of the words in those dictionaries such as who, which, how many etc are the important part of the meaning of the text, because there are the essence of a math task and removing them degraded rather than improved the algorithm. As a result, we decided to create our own dictionary of words, which are safe to be removed from those texts - such as prepositions and names.

The next task related to data cleaning that needed to be done was filtering the data from those that are of particular language – currently in our system we had assignments in both English and Bulgarian and we decided to concentrate our efforts on the Bulgarian ones.

Finally, after all of the pre-processing steps, some records remained too short to be used - after cleaning the data and converting the remainder of the text to lemmas [5] and removing certain stop words, some texts contained less than 3 words - those were removed completely. The final dataset as mentioned contains around 300 records and 24 categories.

2.2 Encoding the text

The classifier cannot use the text directly as input, and therefore a special method needs to be used to map the text document into an appropriate text representation, such as simple word frequency counting, or other [2]. Therefore, after cleaning the texts as described in the previous section, the data is to be vectorized or tokenized and since the sklearn package is used for this task [6] possible ways to do this were explored. It offers several options (classes) - CountVectorizer, TfidfVectorizer, HashingVectorizer. We conducted initial tests with all three of them and decided that TfidfVectorizer would not yield good results on properly classifying the data, because it puts more weight on rare words and considering how short our texts are and the field from which those short texts are, it will put more weight on unimportant part of the assignments. In the next section we are explaining the nature of encoding the data.

2.2.1 CountVectorizer. Machines cannot understand symbols and words, so when dealing with text data, it must be represented in numbers to be understood by the machine [7]. CountVectorizer is a method for converting text to numeric data. To explain how CountVectorizer works, let's have the sentence "I love to go hiking and to go to the movies", a dictionary of unique words is formed and the occurrences of the words in our text are counted – {"I": 1, "love": 1, "to": 3, "go": 2, "hiking": 1, "and": 1, "the": 1, "movies": 1}. More precisely, the way it works is as follows:

- Tokenization: the first step is to break the text into individual words or terms, which is known as tokenization.
- Building the dictionary: the CountVectorizer builds a dictionary from the unique terms in the entire text set. Each unique term is assigned a unique index in the dictionary.
- Count the occurrences of each unique word: For each text in the collection, CountVectorizer counts the number of occurrences of each term in the text and records this information in a matrix. The rows of the matrix represent the texts and the columns represent the terms in the dictionary. The matrix is often referred to as the "document term matrix" (DTM).

- Sparse representation: since most documents contain only a small fraction of the terms in the dictionary, the resulting matrix is usually sparse (contains many null values). This sparse representation is memory efficient and is suitable for further processing in machine learning algorithms.

2.2.2 HashingVectorizer. HashingVectorizer is a feature extraction technique used in natural language processing (NLP) and machine learning. It is specifically used to convert a collection of text documents into a matrix of occurrences of characters. This process is often a crucial step in preparing textual data for machine learning algorithms that require numerical input, such as many classifiers and clustering algorithms.

Unlike traditional vectorizers, such as CountVectorizer or TfidfVectorizer, which maintain a fixed dictionary and assign a unique index to each term in the dictionary, HashingVectorizer uses a hash function to map terms to indexes. HashingVectorizer and CountVectorizer are designed to do the same thing [8]. That is, to convert a collection of text documents into a matrix of character occurrences. The difference is that HashingVectorizer does not store the resulting dictionary of words, which can be both an advantage and a disadvantage [8]. The advantage is that it is very efficient for large amounts of data - if we have a huge number of items in the dictionary, the fact that we don't have to store it along with the object makes the object very small and hence this approach is much more memory-efficient [8]. The disadvantage is that it is not possible to retrieve any token based on a position [8].

In HashingVectorizer, each token is mapped directly to a column position in a matrix whose size is predefined - for example, if you have 10,000 columns in your matrix, each token is mapped to 1 of 10,000 columns [8]. This mapping is done by hashing [8].

This approach has several advantages:

- Fixed size: the output matrix is fixed size, regardless of the size of the input dictionary. This can be useful when working with large data sets where memory efficiency is important.
- Memory efficiency [8]: Since it does not store the term-index mapping in memory, the HashingVectorizer can be more memory-efficient than traditional vectorizers.
- Online training: HashingVectorizer supports online or step-by-step training. You can continuously feed new data to the vectorizer without having to recalculate hash values for the entire dictionary.

However, there are some drawbacks to using HashingVectorizer:

- Lack of reverse transformation [8]: Unlike CountVectorizer or TfidfVectorizer, HashingVectorizer does not provide an easy way to invert the transformation and return the indices to the original terms. This may be a limitation in some applications.
- Risk of collision: since multiple terms can be mapped to the same index due to the use of hash functions, there is a possibility of collision. However, this problem is usually mitigated by using sufficiently large hash spaces.

2.3 Selecting the algorithm

The third step is to select a classification algorithm. To reduce the number of classifiers to test with, an initial investigation was

made into which classifiers were good for classifying short text with few and very diverse examples, such as our data. According to [3] [9] of the traditional methods such as Naive Bayes classifier is likely to perform well. But since a Naive Bayes classifier assumes that the attributes are independent, the assumption is: it will work best when they are truly independent and will degrade under some dependence on the attributes [10]. In this case, even if there were some dependencies between some of the words, it would be more likely to be random, so for our data we assume that this classifier will produce good results (high correct classification). Due to the extensive and thorough research done in [3] SVM was also chosen as possible candidate for classifier. In the next sections we are explaining the nature of both.

2.3.1 Naive Bayes. Before providing a description of a naive Bayes classifier, we need to explain the Bayes theorem. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1, \dots, x_n [6]:

$$P(y|x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

To find the value of $P(y|x_1, \dots, x_n)$ both terms must be evaluated $P(x_1, \dots, x_n|y)$ and $P(y)$. $P(y)$ is easily estimated by calculating the frequency of occurrence of each value y of the result of the function in the training data set. However, the calculation of $P(x_1, \dots, x_n|y)$ can be very difficult if not impossible to estimate. An optimization that makes the computation of $P(x_1, \dots, x_n|y)$ relatively easy without degrading the classifier performance much is to conditionally assume that the attributes x_1, \dots, x_n are independent, therefore we can replace the calculation of $P(x_1, \dots, x_n|y)$ with [6]:

$$P(x_1, \dots, x_n|y) = \prod_{i=1}^n P(x_i|y)$$

Also $P(x_1, \dots, x_n)$ is a constant, given the input, so finally the initial formula is simplified to the formula of Naive Bayes classifier [6]:

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i|y)$$

2.3.2 Support Vector Machine – SVM. Support Vector Machine (SVM) is one of the most popular approaches to supervised machine learning, where no prior knowledge of the problem domain is used. SVM works very well with high dimensional data, avoiding the "curse of dimensionality" [11]. It uses only a subset of training examples - so-called support vectors to represent a solution surface [11]. The SVM solves typically classical classification problems with two classes as shown in Figure 1 [12].

The SVM algorithm searches for the best straight line, which is called the "decision surface" that can separate the data into two classes [12]. Each decision surface corresponds to a linear classifier [14]. Although the classification results of several classifiers may be the same for certain prior data, if other candidate data are considered, the classification results of the two classifiers might be different [12]. The SVM algorithm considers that one classifier in Figure 1 (the one with the straight line) is better than the other (the dotted line) in terms of performance, which is based on the fact that the interval at one classifier is larger than the other, or

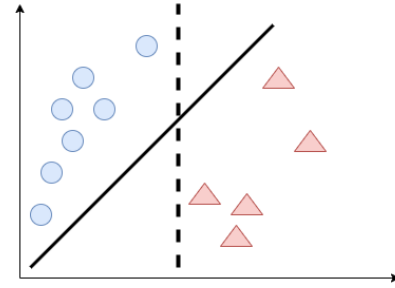


Figure 1: Two classifiers in SVM

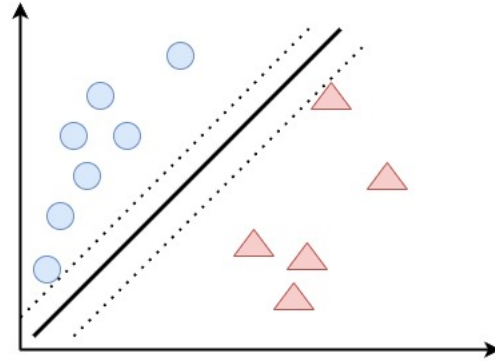


Figure 2: Hyperplane in SVM

in other words, when the direction of the decision surface is kept unchanged and there is no wrong classification, moving the decision surface will find two extreme positions on either side of the original decision surface (crossing this position will result in wrong classification) [12]. The surface between the two dotted lines in Figure 2 is called hyperplane or tolerance limit of the classifier [13] [15]. The line between the two dotted lines is the optimal decision surface provided that the direction of the current decision is preserved [12]. The vertical distance between the two dotted lines is the classification interval corresponding to the optimal solution.

The decision surface with "maximum interval" is the optimal solution to be found by the SVM. The sample of points intersected by the two dotted lines on either side corresponding to the actual optimal solution are the sample support points in the SVM, which is called the "support vector" [12]. Obviously, any line that can correctly partition the examples has an optimal decision surface. However, in some cases it is impossible to completely separate the two types of examples by shifting the surface position in any way [13].

3 TESTS AND ANALYSIS OF RESULTS

After explaining how the data is pre-processed and encoded and searching for a proper algorithm, the last step was to choose classifiers with which to test and assess which has the best accuracy. Based on what we researched in the previous sections and after checking what classes sklearn provides [6], we decided to use RidgeClassifier (Least Squares SVM), RidgeClassifierCV (Least Squares

Table 1: Tests of classifiers using HashingVectorizer encoding

Classifier	% test set	random_state	Accuracy
RidgeClassifierCV	10	22	91.67
RidgeClassifierCV	10	40	95.83
SGDClassifier	10	14	91.67
SGDClassifier	10	40	95.83
LinearSVC	10	14	91.67
LinearSVC	10	22	91.67
LinearSVC	10	40	95.83
LinearSVC	20	14	91.67
PassiveAggressiveClassifier	10	14	91.67
PassiveAggressiveClassifier	10	15	91.67
PassiveAggressiveClassifier	10	22	91.67
PassiveAggressiveClassifier	10	23	91.67

Table 2: Tests of classifiers using CountVectorizer encoding

Classifier	% test set	random_state	Accuracy
RidgeClassifier	10	33	91.67
RidgeClassifier	10	40	95.83
RidgeClassifierCV	10	33	91.67
RidgeClassifierCV	25	40	91.53
MultinomialNB	10	17	95.83
SGDClassifier	10	22	91.67
SGDClassifier	10	23	91.67
SGDClassifier	10	31	91.67
LinearSVC	10	40	95.83
PassiveAggressiveClassifier	10	14	91.67
PassiveAggressiveClassifier	10	22	91.67
PassiveAggressiveClassifier	10	33	91.67
PassiveAggressiveClassifier	10	40	95.83

SVM), SGDClassifier (SVM), LinearSVC (SVM), PassiveAggressiveClassifier and MultinomialNB (Naive Bayes) for the tests, because they use the algorithms that were mentioned in the previous section as promising for our data. We used both HashVectorizer and CountVectorizer for the encoding (one exception is HashingVectorizer - it was not used for Naïve Bayes classifier, because the data cannot be encoded properly with HashingVectorizer to be used with that classifier). This was done to check whether the encoding is crucial for the accuracy of the classification.

For the tests to be comparable, we used the same hyperparameters for all, when applicable – the same regularization strength (1.0), max iterations (1000) and tolerance for stopping criterion (1e-3). For each of the tests the following three steps are performed: data pre-processing, model training, accuracy check. We have conducted tests with test set of 10%, 15%, 20%, 25%, 30% and 35% of the whole dataset and random_state between 10 and 42 with a step 1. The test data showed, the most important step in classifying the data was not the choice of classifier and encoding, but the choice of parameters when splitting the set into a test set and a training set - i.e. what parameters should be passed to train_test_split, namely test_size (what part of the set should be kept as test set), and what

value should get random_state (i.e. an integer that provides the randomness of the split, ensuring reproducibility).

Due to the huge amount of test data (nearly 2000 test records), only the data when the accuracy of the classifier is above 90% is extracted in Table 1 and Table 2. Apparently the way of encoding (HashingVectorizer or CountVectorizer) does not affect the accuracy of the classifier that much, so either of the two encoding can be chosen for the implementation. The other thing that is worth noting is that the best results are achieved with a minimum percentage of data allocated to the test set. This is because we do not have a lot of data (~300 records), and there is also relatively little data from each category (~10-12 records per category), i.e. the records are rather scattered. Once we accumulate more data, this is to be changed.

Also, if we observe the results in Table 2, out of six classifiers that were initially selected for the tests, four of them achieve the maximum value of classifier accuracy, and three of them with the same parameters.

Based on the results we can confirm our theory that because of the scattered data a small test sets will be better. The other parameter that we need to observe and analyse is random_state. We have checked more closely the results for two of the classifiers

with the best accuracy (MultinomialNB classifier and PassiveAggressiveClassifier) with a fixed percentage of the test set at 10% (based on the results show in Table 1 and Table 2, test set of 10% yield the best results for different classifiers). The accuracy of the classifiers varies a lot with the different values of the random_state parameter ([66.67, 95.83]) for both PassiveAggressiveClassifier and MultinomialNB. We already mentioned that this is somewhat expected because the data is small and there are few entries from each category. Those results also showed us that when using different classifiers and depending on the data, it is very important to adjust the parameters when splitting the data into training set and test set.

We also checked the minimum and maximum accuracies of Naive Bayes classifier and PassiveAggressiveClassifier achieved at different percentage of test set (10%, 20%, 25%, 30% and 35%) using different values of random_state ([10, 42]). Those results made it even clearer how important is to choose the right parameters when creating a test and training set. The difference is particularly large – e.g. for a test set of 10% and random_state = 20 the accuracy of the classifier is only 50%, and for test set again 10% and random_state = 17 the accuracy of the classifier is the maximum 95.83.

4 CONCLUSION

Based on the test, almost all the classifiers namely RidgeClassifier, MultinomialNB, SGDClassifier, LinearSVC, PassiveAggressiveClassifier with CountVectorizer encoding, and RidgeClassifier, SGDClassifier, LinearSVC and PassiveAggressiveClassifier with HashingVectorizer encoding have a least one combination of the parameters that yields the best classification value. But looking more closely at the results and according to the characteristics of the classifiers and tests, it is probably a good idea to choose either PassiveAggressiveClassifier or MultinomialNB. For those classifiers we have more stable results, but based on the specifics of the two ways of encoding HashingVectorizer seems like the better choice, which drops MultinomialNB as an option, so finally PassiveAggressiveClassifier with HashingVectorizer for encoding, test set = 10% and proper random_state looks like the best solution for our initial problem.

REFERENCES

- [1] J. Chen, Z. Gong and W. Liu, "A Dirichlet process biterm-based mixture model for short text stream clustering," *Applied Intelligence*, vol. 50, no. 5, pp. 1609-1619, 2020. <https://doi.org/10.1007/s10489-019-01606-1>.
- [2] Tuan Ha and Xiaoying Gao. 2022. Evolving Multi-view Autoencoders for Text Classification. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '21)*. Association for Computing Machinery, New York, NY, USA, 270–276. <https://doi.org/10.1145/3486622.3493969>.
- [3] Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S. Yu, and Lifang He. 2022. A Survey on Text Classification: From Traditional to Deep Learning. *ACM Trans. Intell. Syst. Technol.* 13, 2, Article 31 (April 2022), 41 pages. <https://doi.org/10.1145/3495162>.
- [4] "Stopwords Bulgarian (BG)," [Online]. Available: <https://github.com/stopwords-iso/stopwords-bg>.
- [5] A. Barbaresi, "Simplemma: a simple multilingual lemmatizer for Python," [Online]. Available: <https://github.com/adbar/simplemma/>.
- [6] "Machine Learning in Python," [Online]. Available: <https://scikit-learn.org/stable/>.
- [7] P. Jain, "Basics of CountVectorizer," 24 May 2021. [Online]. Available: <https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c>. [Accessed 28 November 2023].
- [8] K. Ganesan, "HashingVectorizer vs. CountVectorizer," [Online]. Available: <https://kavita-ganesan.com/hashingvectorizer-vs-countvectorizer/>. [Accessed 29 November 2023].
- [9] Zhe Zhang, Zhifeng Wu, and Zhiwei Shi. 2022. An improved algorithm of TFIDF combined with Naive Bayes. In *Proceedings of the 2022 7th International Conference on Multimedia and Image Processing (ICMIP '22)*. Association for Computing Machinery, New York, NY, USA, 167–171. <https://doi.org/10.1145/3517077.3517104>.
- [10] Weicong Guo. 2022. Applications of Logistic Regression and Naive Bayes in Commodity Sentiment Analysis. In *Proceedings of the 2022 4th International Conference on Image, Video and Signal Processing (IVSP '22)*. Association for Computing Machinery, New York, NY, USA, 224–230. <https://doi.org/10.1145/3531232.3531265>.
- [11] Thein Yu and Khin Thandar Nwet. 2020. Comparing SVM and KNN Algorithms for Myanmar News Sentiment Analysis System. In *Proceedings of 2020 6th International Conference on Computing and Data Engineering (ICCDE '20)*. Association for Computing Machinery, New York, NY, USA, 65–69. <https://doi.org/10.1145/3379247.3379293>.
- [12] Yujiao Wang, Haiyun Lin, Chunyu Li, Lina She, Li Sun, and Junwei Wang. 2023. Network Autonomous Learning Monitoring System Based on SVM Algorithm. In *Proceedings of the 2023 10th International Conference on Wireless Communication and Sensor Networks (icWCSN '23)*. Association for Computing Machinery, New York, NY, USA, 98–102. <https://doi.org/10.1145/3585967.3585984>.
- [13] B. Stecanella, "Support Vector Machines (SVM) Algorithm Explained," 22 June 2017. [Online]. Available: <https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/>. [Accessed 3 December 2023].
- [14] Y. Wang, H. Lin, L. She, C. Li, L. Sun and J. Wang. 2022. The Design and Implementation of Information-Based Teaching Skills Training Platform for Primary and Secondary School Teachers. In *International Conference on Education, Network and Information Technology (ICENIT)*. Liverpool, United Kingdom, 43-47. 10.1109/ICENIT57306.2022.00017
- [15] S. Mishra, "Breaking Down the Support Vector Machine (SVM) Algorithm," 29 October 2020. [Online]. Available: <https://towardsdatascience.com/breaking-down-the-support-vector-machine-svm-algorithm-d2c030d58d42>. [Accessed 30 December 2023].

**UNIVERSITY OF RUSE
BULGARIA**

COMPSYTECH'24 CONFERENCE

14-15 JUNE 2024

Scopus[®]



Published by ACM

PAPER SUBMISSION DEADLINE

9 March 2024

<http://www.compsystech.org>

International Conference on Computer Systems and Technologies CompSysTech'24 - 14-15 June 2024, University of Ruse, Bulgaria

Call for Papers:

The International CompSysTech'24 Conference will take place on 14-15 June 2024, at the University of Ruse, Bulgaria.

The Conference Proceedings are published in the ACM International Conference Proceedings Series (ACM ICPS), which has an SJR, and indexed by SCOPUS.

The paper submission deadline is 9 March, 2024. The conference is planned to be held in a mixed mode: on-line and face-to-face.

CONFERENCE TOPICS :

- Computer Hardware.
- Computer Software.
- Application Aspects of Computer Systems and Technologies.
- Educational Aspects of Computer Systems and Technologies (ICT-based innovative educational technologies)

The youth and the Computer Systems and Technologies:

- High-School Students' Section;
- Students' Section;
- PhD Students' Section.

More details can be found in the submission requirement section of the conference web site <http://www.compsystech.org>

We are looking forward to receiving your papers!