Implementing Machine Learning Methods in Searching Processes

Roman Čerešňák
University of Žilina
Žilina, Slovakia
roman.ceresnak@fri.uniza.sk

Karol Matiaško
University of Žilina
Žilina, Slovakia
karol.matiasko@fri.uniza.sk

Adam Dudáš
Matej Bel University
Banská Bystrica, Slovakia
adam.dudas@umb.sk

Abstract— **Methods of machine learning are currently very widespread, popular and are used in number of sectors - whether it is medicine, industry or the transportation. In many industries, machine learning is a factor of improvement which streamlines the process of disease diagnosing, speeds up the process of object identification at airports or eliminates number of errors which may occur in the production process based on previous testing. Machine learning applied to data retrieval processes in various types of databases, whether relational or non-relational ones, can bring more benefits than minimization of data retrieval time or reduction of database server usage. Based on the previous research – which focused on a comparison of the time required to obtain data in relational and non-relational databases - we concluded that it is more appropriate to implement methods and processes of machine learning to non-relational key-value type databases such as MongoDB or DynamoDB. Our proposed solution works with two principles. The first one is the principle of monitoring unfinished commands (operations) and their subsequent transfer to the buffer memory. The second principle is based on definition of the limit at which can machine learning efficiently provide appropriate transfer of the supposedly requested data to the buffer. This action can not only speed up the time required to obtain data, but also provide proposal of data selection operations based on previous queries of the user.**

I. INTRODUCTION

With the arrival of artificial intelligence in the cloud computing, two significant tools have come together in one system - availability of powerful computing resources directly on the network and new sophisticated methods which can find patterns, relationships and correlations even in large amounts of data. Platforms such as Amazon, Azure or Google Cloud make it much easier to work with these systems, as some of the related complex processes allow them to be configured with the use of intuitive user interface.

Artificial intelligence, in the broadest sense of the term, points to the fact, that machines can perform tasks that we generally consider smart. We once thought that something like this could only be achieved in a completely new way of programming, something revolutionary and complex, which is right on the edge of possible knowledge and maybe even further. One of the oldest approaches to creating artificial intelligence is to simulate the functions of the human brain. In 1959, Arthur Samuel introduced work, which contained the term "machine learning," in which he suggested that instead of programmers teaching computers, they should let these machines learn on their own. Paradoxically, machine learning is currently based on a simple set of relatively old statistical algorithms that are performed very quickly and repeatedly and work with large amounts of data. This alone is enough to talk about a revolution. Machine learning can find knowledge, relations, patterns, correlations, estimate unknown values, identify anomalies or classify seemingly unclassifiable in datasets with sizes beyond anything standard tools can process. This turned programming methods upside down: we no longer need to write algorithms to solve problems, the algorithms are written by the computer itself with the use of monitoring of large amounts of data.

Working with large amounts of data and the correct statistical models, it is possible to find patterns and correlations even in data which seem too complex at first glance and their detection without the help of a large computational capacity is more than complicated. Identifying the dependencies and patterns hidden in this data can be used in finding solutions to number of problems faced by various organizations worldwide. As an example, by studying various factors - such as the age and history of client or time and place of payment - indications of suspicious payments in the bank environment can be found. Machine learning can be used to predict future revenue of organizations, estimate events which cause customer to change mobile operators, choose movies for user to watch, determine which machine in factory is inclined to defects soon or uncover any other problem where large amounts of collected data is available.

It is the large amount of data associated with non-relational databases which led several researchers to implement machine learning into multiple types of databases - such as relational and non-real databases. The objective of this action was not only to reduce the number of select, update, insert and delete operations while working with the data in database but also to optimize memory requirements for management of large amounts of data in the mentioned operations. Due to the growing popularity of non-relational databases, researchers decided to apply machine learning to non-relational key-value databases [1]. Since it is versatile and easily adaptable, the MongoDB database was chosen as the basis for this type of applications [2].

We identified several aspects, which – in our opinion – are critical in the data searching process and selection of data, and we summarized them in this research article, which addresses the following points:

- monitoring of unfinished data selection commands in the relational database Oracle and the non-relational database MongoDB,
- providing proposed data selection operations based on previously used queries,

- module, which can be used in transfer of records from relational or non-relational database to the in-memory database Redis.

Presented article is divided into following logical parts - after the introduction, we present related research and articles which deal with the issues researched in this paper and which served as an inspiration for us in designing our solution. In the third section, we introduce the modules and various possible methods in which we implement machine learning into the process of data selection and subsequent transfer of records to the database Redis. We experimentally verify and compare proposed solution with conventionally used methods in the section four. In the last part of the paper, we summarize our solution and present possibilities for future research related to the objective of this article.

II. Related Work

When researching the area of machine learning, we focused on several research articles and other related works. As the most influential and interesting for our research, we include research presented in [3] which plans a flow-based IDS utilizing two machine learning strategies: choice tree J48 and Multilayer Perceptron (MLP). For testing reasons, the authors utilize UNSWNB15 dataset. Authors of the work found out that the utilization of J48 produces better rate of accuracy than straight MLP, which is 0.985 and 0.910, individually. Furthermore, the authors also discovered that expanding the number of layers raises the precision, even though it increases time of computation in the system.

Another research work, which was presented by Muttaqien and Ahmad [4], utilizes highlight determination, clustering and highlight change on the datasets NSL-KDD and Kyoto 2006. Here, clustering is done by executing k-means algorithm whose span of clusters is to be the limit for gathering the information. This strategy is able to improve the classification execution and increase the accuracy of the classification - best results reached on the dataset NSL-KDD is 97.42% and 99.72% on the dataset Kyoto 2006. Research by Thaseen and Kumar presented in [5] focuses on IDS by making a normalization arrange, rank-based chi-square highlight choice and classification with numerous SVMs. The proposed strategy is verified on NSL-KDD and KDD Cup99 datasets. It is illustrated that their strategy is more reasonable for the use with NSL-KDD than KDD Cup99.

In [6] Mukherjee and Sharma explore three correlation-based include-choice strategies applying to include determination issues: correlation-based, information reinforcement and pick-up proportions. In the expansion of the work, they also propose a modern strategy for feature selection utilizing highlight vitality-based lessening strategy to distinguish and after that iteratively reduce less vital highlights. Utilizing the Credulous Bayes classification, they improve performance with the diminished dataset. Authors conclude that diminishing the number of features leads to better execution of tasks.

Akashdeep et al. [7] propose an IDS with highlight choice based on the procurement and relations between data. To choose the highlights, they analyze the securing of the data and relationship comes about. From this information, an unused approach is proposed to sort out highlights which are valuable. For this reason, they utilize nourish forward neural arrange classification in preparing and testing, in expansion to the normalization of the dataset. Compared to that without highlight choice, the utilize of include determination appears way better comes about. Amiri et al. [8] apply two highlight determination strategies to KDD Cup99. They compare the shared information-based highlight choice strategy with relationship coefficient of direct and nonlinear degree for include choice. This strategy has high precision in recognizing Inaccessible to Login (R2L) and Client to Inaccessible (U2R) assaults.

Kasliwal et al. [9] create a crossover show utilizing the Inactive Dirichlet Assignment (LDA) and the Hereditary Calculation (GA). LDA is utilized to distinguish the ideal set of attributes, while GA is utilized to calculate starting scores for wellness esteem assessment to get new features utilized within the classification of KDD Cup99 datasets. Ikram and Cherukuri [10] propose a half breed IDS show with two approaches: Foremost Component Examination (PCA) and Back Vector Machine (SVM). The step to do is to perform parameter selection optimization with PCA on the SVM classifier bit. With optimization of punishment factors and gamma part parameters, this strategy can move forward classification performance and decrease classification time in preparing and testing.

In another paper dealing with machine learning authors [1] apply data mining to classifying those anomaly data. This is based on the facts that there are many data which are not ready for use by a classification algorithm. In addition, that algorithm may use all features which actually are not relevant to the classification target. According to these two problems, we define two steps: pre-processing and feature selection, whose results are classified by using k-NN, SVM, and Naive Bayes. The experimental results show that such pre-processing and combination of CFS and PSO are better to apply to SVM which is able to achieve about 99.9291% of accuracy on KDD Cup99 dataset.

The main objective of another paper deals with machine learning is focused for training and combining intrusion detection datasets. Authors in paper [2] presented a method to train and combine several datasets from semi-structured sources with the MapReduce programming paradigm under MongoDB. It aims to increase the intrusion detection rates. In their work, they are focus on KDD99, DARPA 1998 and DARPA 1999 dataset and with the big data technique MapReduce in MongoDB: First, authors selected the most pertinent attributes and eliminate the redundancies from the previous datasets. Then, authors merged them vertically into the same collection. Finally, they analyzed the dataset they used a Bayesian network as K2 algorithm implemented in WEKA.

Despite the achieved results, which were presented in works of other authors and our previous work [11], [12], [13], [14], we decided to implement a solution which consists of combination of several modules. This combination of modules works on the basis of following three tasks:

- replacing the computationally costly workflow controlled by the machine learning with a faster workflow similar to the one used with queries,
- replacing the complexity of tight models and supervised learning with a single multipurpose system and queries similar to SQL and NoSQL databases,
- maintaining effective performance and quality of prediction compared to supervised learning.

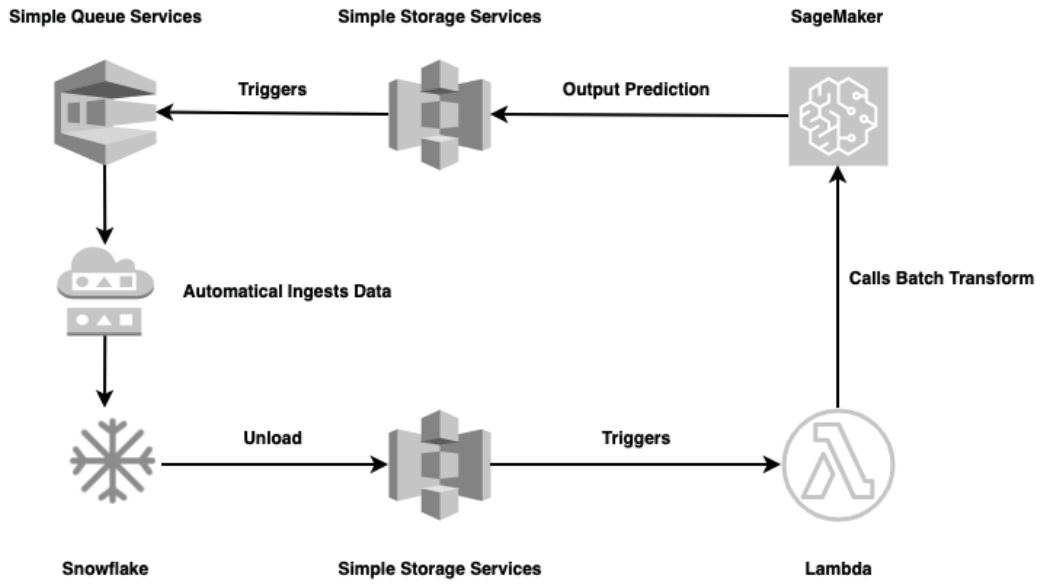

Fig. 1. Diagram of proposed architecture

III. Architecture For Data Searching With the Use of Machine Learning

To implement and properly operate machine learning in the context of data searching, we created an architecture, which is presented in the Fig. 1 and work as follows:

- The client enters the data into S3 with the use of the required structure which triggers Lambda function.
- SageMaker *Batch Transform* job is called to preprocess loaded dataset for further processing in proposed model.
- Machine learning predictions are created and sent back onto the S3 bucket.
- SQS is set up on given S3 bucket to auto-ingest the predicted result onto Snowflake
- Once the data lands onto Snowflake, Streams and Tasks are called.

Due to the need for efficient record processing, we created a function which is used for transferring data from file to S3 (Simple Storage Services). For a simpler implementation, we created a procedure which moves the records and then stores them in two tables of relational database MySQL.

These tables are shown in Fig. 2 - specifically the tables *order_cancellation* and *prediction_status*, which are filled using the mentioned procedure controlled by machine learning. The whole principle is covered by Snowflake.

In order to call *Batch Transform*, we need to create an input table which contains data for the model and mandatory fields, *predictionid* which is the *uuid* for the task, *record_seq* which is unique identifier, for reach input row, a NULL value is stored in the prediction column – this column is targer of interest.

The data we tested are presented in the Table I and serve as samples, which were then moved from the S3 service to the non-relational database MongoDB. Since we use number of services in the cloud computing solution from Amazon, where there is no direct support for the MongoDB, we have chosen a direct

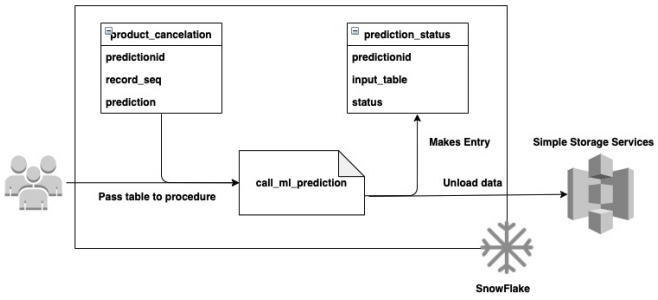

Fig. 2. Unloading data onto S3

substitute for these purposes and that is the non-relational database DocumentDB, which has almost identical properties.

As can be seen in the Table I, the values we used for our testing and monitoring of proposed model contains 6 columns, while the value in the prediction column is initially set to the unknown value NULL. This value indicates that no prediction has been performed on the given data yet. Modification of this column is focus of the following steps of proposed method – these NULL values will be adjusted on the basis of a correctly or incorrectly performed prediction, created by the user in the process of obtaining values using the proposed architecture.

The *call_ml_prediction* Stored Procedure takes in a user-defined job name and input table name. Calling it will unload the file (using *predictionid* as the name) onto S3 bucket in the /input path and create an entry in the *prediction_status* table. From there, Batch Transform will be called to create prediction based on the input data.

In order to be able to call the procedure, we wrote a command which will call the SQL script, which can be found at the following link:

https://github.com/romanceresnak/fruct29/blob/main/unload2s3_storedproc.sql

TABLE I. SAMPLE DATA USED IN PROPOSED MODEL

PREDICTIONID	PREDICTION_SEQ	PREDICTION	ORDER	LEAD_TIME	ARRIVAL_DATA_YEAR	ARRIVAL_DATE_MONTH
46c23423-23423-234-4563456	1	NULL	Car	23	2020	May
46c23723-34423-234-4333456	2	NULL	Bike	572	2019	June
46c25623-28983-634-4535676	3	NULL	PC	324	2021	September
46c28883-23423-774-4885456	4	NULL	Mobile	32	2018	June
46c23423-23423-234-4563456	5	NULL	Cloud	55	2017	July
46g44423-23423-986-4563456	6	NULL	Mouse	234	2019	December
46c20023-23423-234-4568888	7	NULL	Keyboard	753	2020	February
46c28883-77723-999-4598456	8	NULL	Server	876	2020	March
46c44423-23423-234-4563456	9	NULL	Router	345	2021	April

This script directly manages the Snowflake, which calls the procedure in case of a query for individual data and then updates the values in the tables.

While researching the area and experimenting with proposed method some undesirable behavior emerged - in the case of multiple users who requested the same data, the individual commands were blocked. This fact caused certain delay for one of the users. This led us to decision, that it won't be possible to run more requests concurrently. Also, for system simplicity and more efficient request management, we have ensured that only one file is loaded on S3, but Batch Transform can process multiple input files simultaneously as shown in Fig. 3.

TABLE II. PREDICTION STATUS TABLE

PREDICTIONID	PREDICTION_JOB	INPUT_TABLE	STATUS
46c28883-77723-999-4598456	2021-27-02	ORDER_CANCELLATION	SUBMITTED

In order to create individual predictions, we need to secure correct management of data and their subsequent processing.

Once the data is read and moved to S3, the Lambda function is called – this function triggers the reading of data and their subsequent processing. Due to the frequent loading of data from file, we decided to subsequently transfer this data to the non-relational database DocumentDB, mainly due to easier data management and data searching possibilities.

Although in many situations the data prediction was fast, we decided to implement the Database module into the architecture shown in the Fig. 3. The mentioned database module is responsible mainly for data transferring. During the prediction, the unfinished command (operation), which is being written by user, is obtained from the non-relational database and then, using a simple script the records are transferred to the Redis database. The script is stored on the following link:

https://github.com/romanceresnak/fruct29/blob/main/cache.js

The created module, which is presented on the Fig. 4, allows us to acquire data even faster. Other than that, this module takes care of management of data. The prediction of the data searching command was relatively inaccurate in the initial steps, and the record transferred from the non-relational database to the memory database (Redis) used to be inaccurate in the initial phase.

For management reasons, the proposed architecture takes care of record transferring in the following way:

- The algorithm follows the currently written command:
  - In the case it has already been found in previous learning cycles, that the predicted command is correct, then the record is moved to the in-memory database.
  - If the given command has not yet occurred during learning, then the data with the largest volume are transferred to the in-memory database (the data with the largest volume are understood as data with the largest number of tables used by user, eg. records consisting of data from 5 tables).
- Transferred records are used to provide data for reference to the SageMaker.
- However, if prediction fails or is inaccurate, one of these two situations occurs:
  - If a situation occurs that the prediction failed and the records in in-memory database are not sufficient, the data is deleted from the memory and records are retrieved from the non-relational database.
  - In other case, the data is sufficient, and system reduces based on the new prediction - not in a non-relational database, but directly in the in-memory database.
- In this way, we obtain values which are transferred to the file, and at the same time the records in the table are updated for further learning.

In such case, that the algorithm obtains the correct data, this dataset is provided to the SageMaker file. After transfer of the data, the Lambda function, which is not set to the default value of 5 minutes, is called. The function is set to monitor the action caused by the filling of the SageMaker file. Subsequently, we create a file, which contains a copy of the data. This file is provided directly to the user.

In order to make management of the individual process less demanding, we decided to share all used files and data not only in the users' process, but also in the processes of other participants by introducing files into S3, which can be used in other predictions.

For the proper functioning of the architecture and subsequent transfer of the records, we created two files, which are run automatically based on the process confirmation. These files can be found at:

https://github.com/romanceresnak/fruct29/blob/main/lambda_call_batch_transform.py

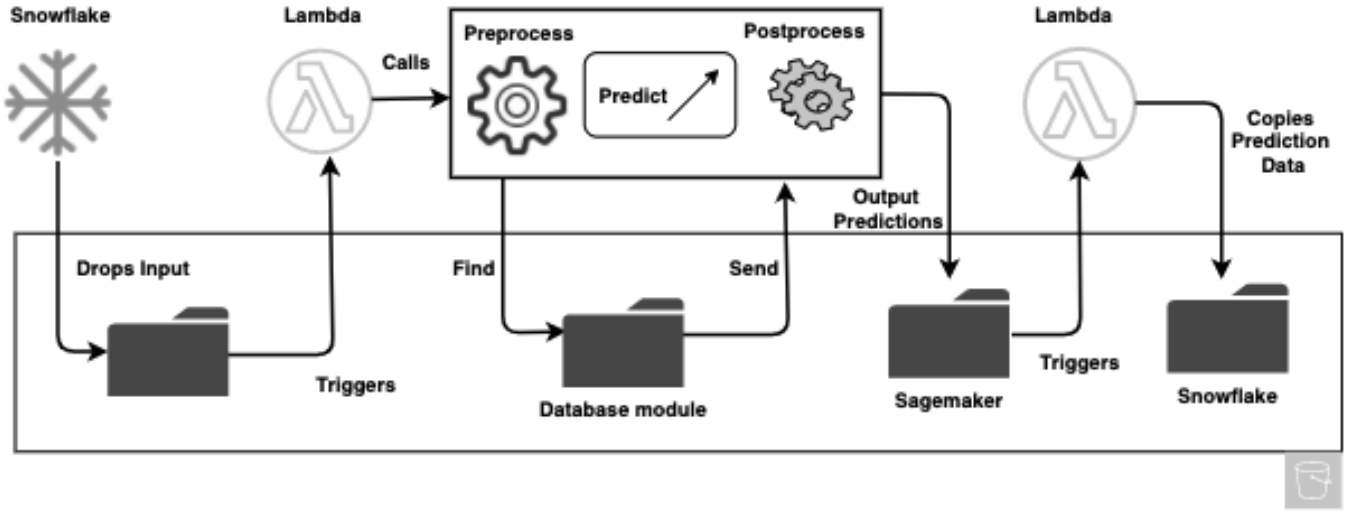

Fig. 3. Data flow diagram of proposed architecture

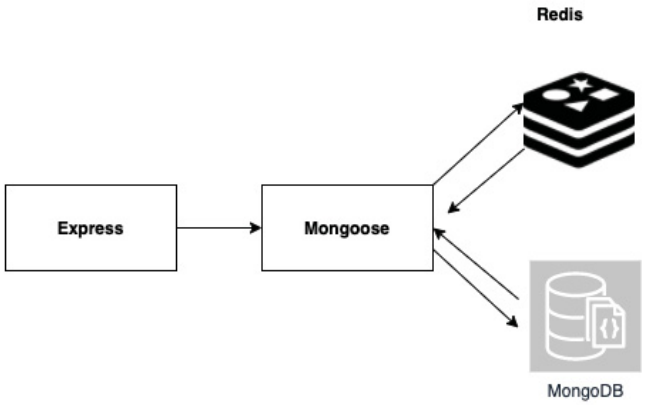

Fig. 4. Design schema of the proposed database module

Once Batch Transform completes, it outputs the result as a *.csv.out* in the /sagemaker path. Another Lambda gets fired which will copy and rename the file as .csv to the /snowflake path where SQS is setup for Snowpipe auto-ingest.

Once the data is dropped onto the /snowflake path, it is inserted into the prediction_result table via Snowpipe. For simplicity, since SageMaker Batch Transform maintains the order of the prediction, the row number was used as the identifier to join to the input table. We did the postprocessing step within Batch Transform itself.

The script needed to perform the above-mentioned operations is stored on GitHub, specifically at the following address:

https://github.com/romanceresnak/fruct29/blob/main/snowflake2sagemaker_snowpipe.sql

Data streaming and running tasks were among the subprocesses which were necessary to secure during the process. For this reason, we have created a stream, presented on the Fig. 5, in which *prediction_result* fills *prediction_result_stream* after Snowpipe delivers the data. This stream, specifically the system$stream_has_data 'prediction_result_stream, is used to schedule *populate_prediction_result* tasks to call the *populate_prediction_result* stored procedure to fill prediction data in the *hotel_cancellation* table only if there is a stream. Unique identifier *predictionid* is set to be relational variable of the task.

To update the values, we had to adjust the procedure, which updates the values based on the correct or incorrect prediction. The modified procedure can be seen at the following link:

https://github.com/romanceresnak/fruct29/blob/main/snowflake2sagemaker_populate_prediction_result.sql

After applying and implementing the procedure which ensures correct prediction, our values have also changed. In the table I, prediction was equal to NULL, which means that the prediction was not performed or was interrupted. After applying our modified procedure, the data in the Table II, specifically the prediction column, was updated. We present the modified table and its values in Table III.

At the end of the task and when *populate_prediction_result* is completed using the system task session variable, the next *update_prediction_status* task updates the prediction status from "Sent" to "Completed" (see Table IV). This completes the whole process.

IV. Experiments On the Architecture For Data Searching With the Use of Machine Learning

Even in design and implementation of the proposed architecture, we needed to secure and effectively encounter several situations which would be able to significantly influence and distort the correctness of the achieved results in a negative way. Based on these conditions, we started the process with the same amount of data and with the default settings of not only the entire cluster (while maintaining the "free tier" mode) but also the basic settings of database servers, specifically DocumentDB and Redis with settings recommended directly by Amazon.

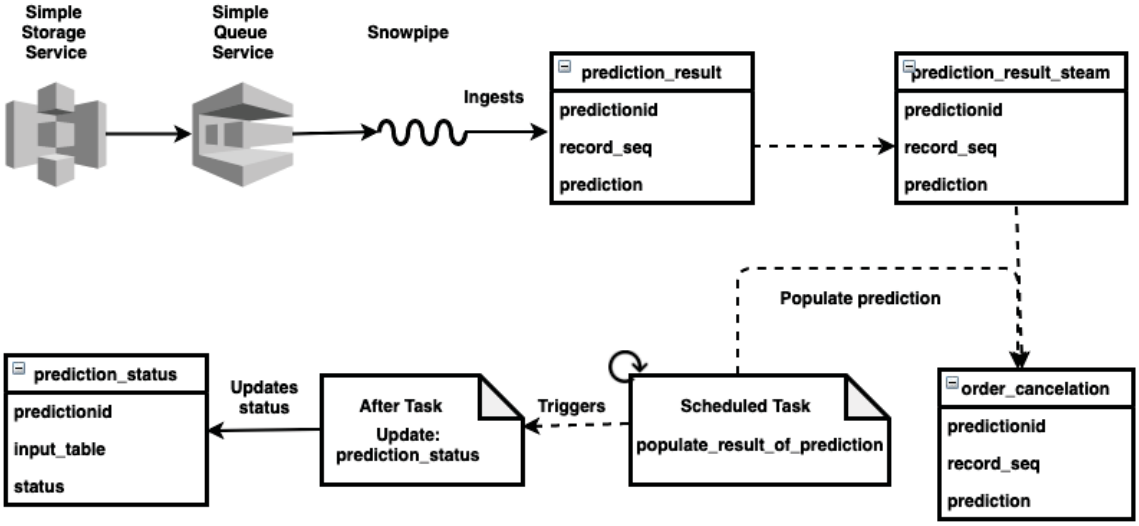

Fig. 5. Flow diagram of piping the data into Snowflake

TABLE III. THE RESULT FROM BATCH TRANSFORM

PREDICTIONID	PREDICTION_SEQ	PREDICTION	ORDER	LEAD_TIME	ARRIVAL_DATA_YEAR	ARRIVAL_DATE_MONTH
46c23423-23423-234-4563456	1	1	Car	23	2020	May
46c23723-34423-234-4333456	2	1	Bike	572	2019	June
46c25623-28983-634-4535676	3	1	PC	324	2021	September
46c28883-23423-774-4885456	4	1	Mobile	32	2018	June
46c23423-23423-234-4563456	5	0	Cloud	55	2017	July
46g44423-23423-986-4563456	6	1	Mouse	234	2019	December
46c20023-23423-234-4568888	7	1	Keyboard	753	2020	February
46c28883-77723-999-4598456	8	1	Server	876	2020	March
46c44423-23423-234-4563456	9	0	Router	345	2021	April

TABLE IV. PREDICTION STATUS VALUE

PREDICTIONID	PREDICTION_JOB	INPUT_TABLE	STATUS
46c28883-77723-999-4598456	2021-27-02 Cancelation Prediction	ORDER_CANCELLATION	Completed

The results of experimental evaluation of the proposed architecture for data searching with the use of machine learning are presented in the Fig. 6.

While implementing our architecture for selecting data from a non-relational database, it was clear that the process would be more time-effective, but we did not know how effective this process would be. As can be seen in the Fig. 6, the overall time of computation of the problem is lowered by approximately 60%. Even though this improvement is satisfying, we improved the process even more by introducing proposed database module (see Fig. 3).

The advantage of the database model was not significant at the beginning of experimental testing since our data management between the in-memory and the non-relational database was not managed by the prediction management. This caused latency in management of records. After the implementation of data management for machine learning, the server utilization was reduced, and subsequently computational time of the problem was reduced even more – to approximately 34% of computational time compared to the time of pure machine learning implementation. Therefore, proposed approach reduced computational time of the problem to 13,5% of its original duration.

It is necessary to mention one fact when applying this method. The time reduction we obtained is significant compared to conventional methods, but the usage and management of the server increased from 10% to a value of approximately 53% while maintaining the "Free tier" account.

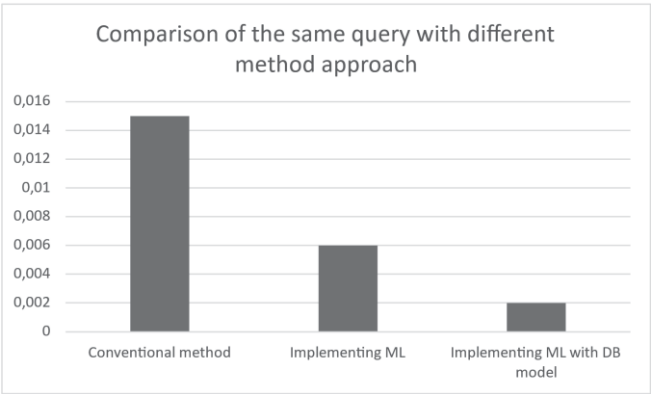

Fig. 6. The result of comparison of various methods for the same query

During the testing, we focused on the influence of change of programming language while the algorithms were running. Many algorithms were programmed in the python language, which allowed us to quickly access the necessary functionalities, but unfortunately it also had drawback of lower computational speed. When rewriting source codes from python to GO (golang), the speed of computation increased as much as 12 percent. The problem that has arisen with rewriting source codes is the length of implementation e.g., the number of lines of script originally written in the python language increased by almost 79 percent when rewritten in Golang, which caused a significant prolongation of the development of the mentioned application.

On the other hand, by rewriting the code, server load increased by 13 percent – this is caused by GOs' ability to use multiple computational units (cores) at the same time, which is not implemented as efficiently in the Python. We also note that the results we measured may differ diametrically from the results that will be obtained on a server with a larger number of cores.

V. Conclusion

Currently, the growing volume of data causes significant problems in various industries and areas of interest. A large amount of data must be stored efficiently for system to be able to capture the demand for data in a relatively short time and then provide the data to the user. Due to this fact, we applied machine learning to the process of obtaining data from the non-relational database MongoDB, which is considered by many researchers to be the most universal non-relational database. Since we designed and implemented a module which would also meet the versatility of relational databases, we also created a method which could work with relational database Oracle.

Use of machine learning in proposed architecture caused significant time reduction during the initial implementation of the process. Predicted commands were not accurate enough in the first steps, but after successful data acquisition, the efficiency of the implementation process manifested itself and caused the duration of computation of process to be reduced.

The proposed method works efficiently and reliably in two steps. The first step is to monitor the unfinished command in the database environment, with the method transferring data from the requested database to the in-memory database and thus speeding up the data acquisition process. If the user uses clauses which reduce the amount of data process works with, the method does not apply the condition in the used database, but in the cache memory. The second step is based on the possibility of providing user with commands and operations based on previous queries. Even before confirming the proposed command records of data are moved into the cache. Based on the achieved results, we can clearly state that the process proposed in this research article can always obtain the requested data faster than with the use of conventional methods.

The implementation of machine learning in search processes has the potential to further improve not only the record search process itself, but many factors also suggest that a similar implementation could help with data update processes. Our future goal is to implement the created module for other types of non-relational databases and to create a general module that will provide the functionality we created using a simple access point.

Acknowledgment

This work was supported by Grant System of University of Zilina No. 1/2020. (8056).

The research was partially supported by the grant of The Ministry of Education, Science, Research and Sport of Slovak Republic - Implementation of new trends in computer science to teaching of algorithmic thinking and programming in Informatics for secondary education, project number KEGA 018UMB-4/2020.

References

[1] T. Ahmad and M. N. Aziz, "Data preprocessing and feature selection for machine learning intrusion detection systems," ICIC Express Lett., vol. 13, pp. 93–101, Jan. 2019.

[2] E. Marwa and F. Jemili, "Using MongoDB Databases for Training and Combining Intrusion Detection Datasets," 2017, pp. 17–29.

[3] L. Van Efferen and A. M. T. Ali-Eldin, "A multi-layer perceptron approach for flow-based anomaly detection," in 2017 International Symposium on Networks, Computers and Communications (ISNCC), 2017, pp. 1–6.

[4] I. Z. Muttaqien and T. Ahmad, "Increasing performance of IDS by selecting and transforming features," in 2016 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), 2016, pp. 85–90.

[5] I. Sumaiya Thaseen and C. Aswani Kumar, "Intrusion detection model using fusion of chi-square feature selection and multi class SVM," J. King Saud Univ. - Comput. Inf. Sci., vol. 29, no. 4, pp. 462–472, 2017.

[6] S. Mukherjee and N. Sharma, "Intrusion Detection using Naive Bayes Classifier with Feature Reduction," Procedia Technol., vol. 4, pp. 119–128, 2012.

[7] Akashdeep, I. Manzoor, and N. Kumar, "A feature reduced intrusion detection system using ANN classifier," Expert Syst. Appl., vol. 88, pp. 249–257, 2017.

[8] F. Amiri, M. Rezaei Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," J. Netw. Comput. Appl., vol. 34, no. 4, pp. 1184–1199, 2011.

[9] B. Kasliwal, S. Bhatia, S. Saini, I. S. Thaseen, and C. A. Kumar, "A hybrid anomaly detection model using G-LDA," in 2014 IEEE International Advance Computing Conference (IACC), 2014, pp. 288–293.

[10] S. Thaseen, "Improving Accuracy of Intrusion Detection Model Using PCA and optimized SVM," J. Comput. Inf. Technol., vol. 24, pp. 133–148, Jun. 2016.

[11] M. Kvet, V. Šalgová, M. Kvet and K. Matiaško, "Master Index Access as a Data Tuple and Block Locator " in Conference of Open Innovation Association, FRUCT, 2019, pp. 176–183.

[12] M. Kvet, "Data Distribution in Ad-hoc Transport Network", Proceedings of the International Conference on Information and Digital Technologies 2019, pp. 275-282 – ISBN 978-172811401-9

[13] M. Kvet, "Relational data index consolidation", 28th Conference of Open Innovations Association FRUCT, FRUCT 2021; Virtual, Moscow; Russian Federation, - ISBN 978-952692444-1

[14] J. Škrinárová, L. Huraj, V. Siládi, "A neural tree model for classification of computing grid resources using PSO tasks scheduling", Neural Network World, Volume 23, Issue 3, 2013, Pages 223-24