Applied Computer Science, vol. 18, no. 1, pp. 89–98
doi: 10.35784/acs-2022-7

Submitted: 2022-02-19 | Revised: 2022-03-21 | Accepted: 2022-03-24

Keywords: source code, binary classification, text classification, AutoML

Marcin BADUROWICZ [0000-0003-2249-4219]*

DETECTION OF SOURCE CODE IN INTERNET TEXTS USING AUTOMATICALLY GENERATED MACHINE LEARNING MODELS

Abstract

In the paper, the authors are presenting the outcome of web scraping software allowing for the automated classification of source code. The software system was prepared for a discussion forum for software developers to find fragments of source code that were published without marking them as code snippets. The analyzer software is using a Machine Learning binary classification model for differentiating between a programming language source code and highly technical text about software. The analyzer model was prepared using the AutoML subsystem without human intervention and fine-tuning and its accuracy in a described problem exceeds 95%. The analyzer based on the automatically generated model has been deployed and after the first year of continuous operation, its False Positive Rate is less than 3%. The similar process may be introduced in document management in software development process, where automatic tagging and search for code or pseudo-code may be useful for archiving purposes.

1. INTRODUCTION

In the community of software development, there is a huge network of applications and websites provided for knowledge exchange. While aspiring software developers may use the community help to search for answers for their questions, the more experienced ones could respond and provide feedback but also take part in discussions about software architecture and more advanced topics. These discussions are highly technical and full of jargon, but also the requirement is to show snippets of source code in various programming languages to achieve proper comments.

Two problems are arising – to properly format the source code published on the website, it must be properly marked as a source code, using either the HTML5 code element or the triple backtick element in the Markdown system. Inability to properly mark the code will render the code snippet hard to read by other software developers, as presented in figure 1 for example. On the other hand, the misformatted code snippet may be completely impossible to understand because it lacks proper indentation, e.g., in the case of the Python programming language.

* Lublin University of Technology, Faculty of Electrical Engineering and Computer Science, Department of Computer Science, Poland, m.badurowicz@pollub.pl

In the last decade, both Machine Learning (ML) and Natural Language Processing have been applied in the study of the source code (Ugurel, Krovetz & Giles, 2002), and the problem of classification of programming languages has been also broadly discussed (Van Dam & Zaytsev, 2016), using Neural Networks (Gilda, 2017), Bayesian learning (Khasnabish et al., 2014) or Multinomial Naïve Bayes (Alreshedy et al., 2018), Convolutional Neural Networks (CNN) (Ohashi & Watanobe, 2019) and Neural Text Classification (LeClair, Eberhart & McMillan, 2018) or even, alternatively the usage of Speech Recognition techniques (Madani et al., 2010). However, all of the presented similar works are mostly concentrating on differentiating between different programming languages than on a problem of differentiating between source code and natural text, however some of the presented concepts may be useful for the presented problem.

Two already available libraries for programming languages are also available right now, pygments for Python (Pygments – Python Syntax Highlighter, n.d.) and github-linguist (Linguist, n.d.) for Ruby, also suited for recognizing a programming language, but able to return "text" classification in case of pygments and "nil" result in Ruby when the text does not seem to be a source code.

```
#include <iostream>

int main() {

  int n = 0;
  std::cin >> n;

  std::string dummy;

std::getline(std::cin, dummy); // ***

for(int i = 0; i < n; ++i) {
std::string line;
std::getline(std::cin, line);
std::cout << '[' << line << ']' << std::endl;
}
}
```

Fig. 1. Example of wrongly formatted source code on the website

Source code understanding could be also used in other contexts, e.g., training ML systems to understand natural language to produce output in a given programming language, starting with CoNaLa dataset (Yin et al., 2018) and currently available commercial products like GitHub Copilot (GitHub Copilot – Your AI Pair Programmer, n.d.).

The problem presented in this paper is a little bit different, as there is no need to identify the programming language, only a need to differentiate between a programming language and the natural text. However, this "natural text" may include specific jargon, names of variables, or even keywords of a programming language. Additionally, the system should be able to identify programming language for any kind of given language, both with more natural-looking syntax e.g., SQL and Python, as well as the ones with a very complex or mathematical-like syntax, e.g., Prolog and Haskell.

The overall document management, especially in software development may also benefit from this kind of solution, where some parts of the pseudo-code or fragments of real life implementation may be properly detected and marked in the documentation of the software development process for easier analysis and archiving purposes.

The final problem is that the system was proposed for one of a Polish software development discussion forum (4programmers.net, 2000), where jargon and overall discussion texts are very different from that in the anglosphere. The second part of this same problem is that in case of internet communication the "natural text" may also include abbreviations, internet acronyms and emotes/emoji.

The author proposed an automated system analyzing the publications on the website and providing feedback if there is a source code block found, which was not marked properly, resulting in the wrong presentation on the website, which will be tested on the website and deployed if successful, working under a set of conditions:

- It will be automatically tracking new threads and discussions on the website,
- It should allow recognition if a single line of text is a programming language snippet or just a text,
- Text may include small fragments of programming syntax,
- Code may be in any kind of programming language,
- Text will be mostly in Polish, but may include English loanwords and jargon,
- After final deployment, if there will be an unmarked code fragment recognized, the system should notify the original author and/or moderators of the website.

2. THE PROPOSED SOLUTION

To solve the first problem, to recognize if the given fragment of a text is a source code or not, a few solutions were considered:

- Using an database for keywords of different programming languages, and a scoring system,
- A scoring system based on lack of whitespace characters between words and a huge number of characters normally not used in a text (e.g., square brackets, semicolons at the end of the line, and similar),
- A statistical system using the average metrics of natural language (length of the word, length of the sentence) in contrast,
- Naïve Bayes system trained on keywords of different programming languages,
- A Machine Learning-based binary classification for text.

After reviewing the similar papers mentioned earlier and similar discussions on other software development websites, the author decided to try to use the Machine Learning approach.

The author has added one more requirement – it would be best if the model could be generated automatically, without user intervention, without manual planning of Artificial Neural Network (ANN) structure (Kulisz et al., 2021) and fine-tuning (Sobaszek, Gola & Kozłowski, 2020).

Such an approach forced to use the automated machine learning (AutoML) solutions, where, provided with only dataset and the definition of the problem (in the case of this paper – text binary classification), the system can automatically generate the ML model to be used in created software, to reduce time and complexity of crafting the solution (He, Zhao & Chu, 2021).

The multi-label classification of the text has been discussed (Wever et al., 2021) as a more challenging problem, but the binary classification is already available in commercially available open-source ML toolkit for the .NET platform, ML.NET (Ahmed et al., 2019). AutoML is also available for modelling and prediction problems (Machrowska et al., 2020), e.g. using LSTM method (Szabelski, Karpiński & Machrowska, 2022). In this paper the text classification will performed, but classification of images (Kłosowski et al., 2021) is also available with such toolkits.

2.1. The basic and final datasets

Because the website in question is providing the Application Programming Interface (API) for getting the content of threads and single posts in the discussion forum module, the basic dataset was prepared by downloading several newest posts from the system using created software in the C# programming language and manually categorizing them into a tab-separated (TSV) file, where each line of the file consisted of two columns – first was deciding if the line is code or text by setting 0 (not code) or 1 (code) value, the second was a single paragraph of text extracted from the discussion. The data has been manually checked to include also longer texts with fragments of the source code inside. Finally, the basic dataset was consisting of 145 examples of text and 96 examples of code, totaling 241 lines in the TSV file, about 53 KB of text. An example of the format of the dataset is presented in figure 2. The source code in the dataset was including C#, SQL, and XML languages.

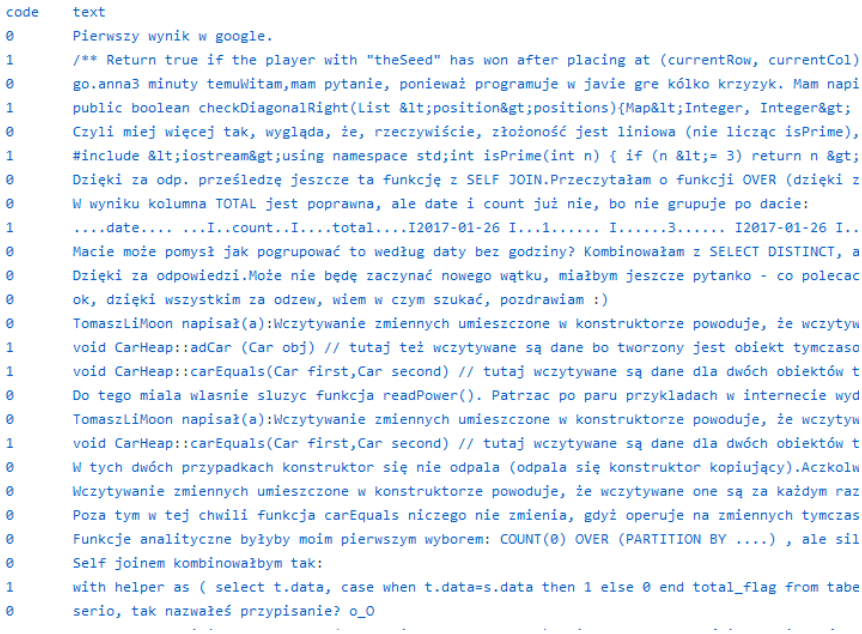
The first, basic, dataset, was used for evaluation of AutoML pipeline available in ML.NET as well as already available libraries, mentioned earlier: pygments and linguist.

Pygments library marked 52 out of those 96 source code samples as text, wrongly classifying 54.16% of examples, marking it completely unreliable in this given task. The linguist's results were also completely unsuitable for the task, as it couldn't properly classify 95 out of 96 code fragments in the basic dataset except for one line consisting of the fragment of an XML document.

It must be noted that both libraries are better suited to classify the programming language on the whole file or repository, not on a single-line code snippet. Those particularly poor results however forced the author to solve the problem using a solution crafted specifically to the problem.

The first AutoML generated model was using the AveragePerceptronOva trainer and reported an accuracy of 95.83%, which was already a very promising result. The preliminary trained model (on the basic dataset) was then used to provide a final dataset.

The web scrapping system has been improved to get more posts and pre-classify them using the ML model acquired in the first step. Then, the dataset labeling has been fixed manually.



code	text
0	Pierwszy wynik w google.
1	/** Return true if the player with "theSeed" has won after placing at (currentRow, currentCol)
0	go.anna3 minuty temuWitam,mam pytanie, ponieważ programuje w javie gre kólko krzyzyk. Mam napi
1	public boolean checkDiagonalRight(List <position>positions){Map<Integer, Integer>
0	Czyli miej więcej tak, wygląda, że, rzeczywiście, złożoność jest liniowa (nie licząc isPrime),
1	#include <iostream>using namespace std;int isPrime(int n) { if (n <= 3) return n >
0	Dzięki za odp. prześledzę jeszcze ta funkcję z SELF JOIN.Przeczytałam o funkcji OVER (dzięki z
0	W wyniku kolumna TOTAL jest poprawna, ale date i count już nie, bo nie grupuje po dacie:
1	....date.... ...I..count..I....total....I2017-01-26 I...1..... I.....3..... I2017-01-26 I..
0	Macie może pomysł jak pogrupować to według daty bez godziny? Kombinowałam z SELECT DISTINCT, a
0	Dzięki za odpowiedzi.Może nie będę zaczynać nowego wątku, miałbym jeszcze pytanko - co polecac
0	ok, dzięki wszystkim za odzew, wiem w czym szukać, pozdrawiam :)
0	TomaszLiMoon napisał(a):Wczytywanie zmiennych umieszczone w konstruktorze powoduje, że wczytyw
1	void CarHeap::adCar (Car obj) // tutaj też wczytywane są dane bo tworzony jest obiekt tymczaso
1	void CarHeap::carEquals(Car first,Car second) // tutaj wczytywane są dane dla dwóch obiektów t
0	Do tego miala wlasnie sluzyc funkcja readPower(). Patrzac po paru przykladach w internecie wyd
0	TomaszLiMoon napisał(a):Wczytywanie zmiennych umieszczone w konstruktorze powoduje, że wczytyw
1	void CarHeap::carEquals(Car first,Car second) // tutaj wczytywane są dane dla dwóch obiektów t
0	W tych dwóch przypadkach konstruktor się nie odpala (odpala się konstruktor kopiujący).Aczkolw
0	Wczytywanie zmiennych umieszczone w konstruktorze powoduje, że wczytywane one są za każdym raz
0	Poza tym w tej chwili funkcja carEquals niczego nie zmienia, gdyż operuje na zmiennych tymczas
0	Funkcje analityczne byłyby moim pierwszym wyborem: COUNT(0) OVER (PARTITION BY ....) , ale sil
0	Self joinem kombinowałbym tak:
1	with helper as (select t.data, case when t.data=s.data then 1 else 0 end total_flag from tabe
0	serio, tak nazwałeś przypisanie? o_O

Fig. 2. Dataset fragment example

In total, the final dataset consists of 1974 lines of text (448 KB in total), consisting of 83% of regular text (which may include small fragments of code snippets) and 17% (335 lines) of pure code snippets. This dataset was including more programming languages – C#, C and C++, Java, JavaScript, HTML, XML, SQL, and Python, where all of them have their own characteristics.

2.2. The ML model generation using AutoML

The AutoML solution included in the ML.NET platform is based on: definition of the problem and the time given the algorithm to test for different results – during a limited time, the software is testing various algorithms, trying to find the best solution to a given problem. In the described situation, the problem is "binary-classification" and 3 runs have been tested, on Core i7-6600U machine without GPU acceleration: with 30-, 60- and 180-seconds time limits.

The ML.NET is grading the solution using three indicators: Accuracy, Area Under Precision-Recall Curve (AUPRC), and FPR Area Under the Curve (AUC).

For 30 seconds run result, the system tested two classifiers, SDCA Logistic Regression and Averaged Perceptron, both suited for binary classification problem. The indicators marking the tested solutions are presented in figure 3.

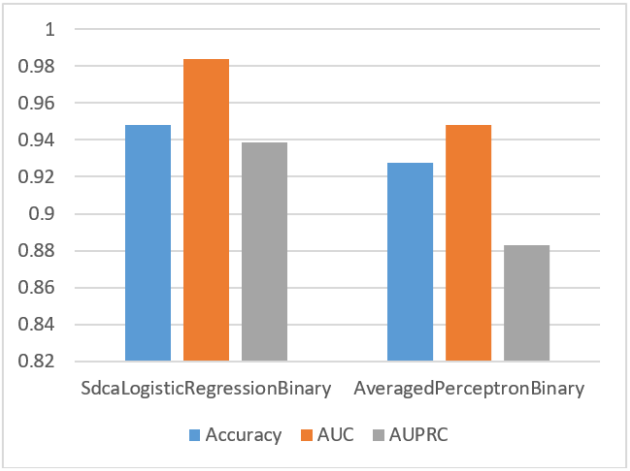


Fig. 3. Results for 30 seconds run

For longer runs, more algorithms have been tested, as presented in figures 4 and 5, including SVM and GBM.

Finally, the best solution returned from the ML.NET autotraining subsystem was based on SDCA Logistic Regression with an accuracy of 95.72%, AUC 95.65%, and AUPRC 88.01% returned after the 180-second run.

Longer autotrain sessions were also tried, but there was no better result returned from the AutoML system than SDCALogisticRegressionBinary.

The final step of the AutoML subsystem is generating a usable trained model for consumption for the .NET platform. The result is a ZIP file consisting of a model and a sample code fragment showing the example usage of the code. The code and the trained model were later used in the deployment step of the proposed solution.

Finally, the system trained on the final dataset has been tested on the smaller, basic dataset, achieving an accuracy of 94.79%.

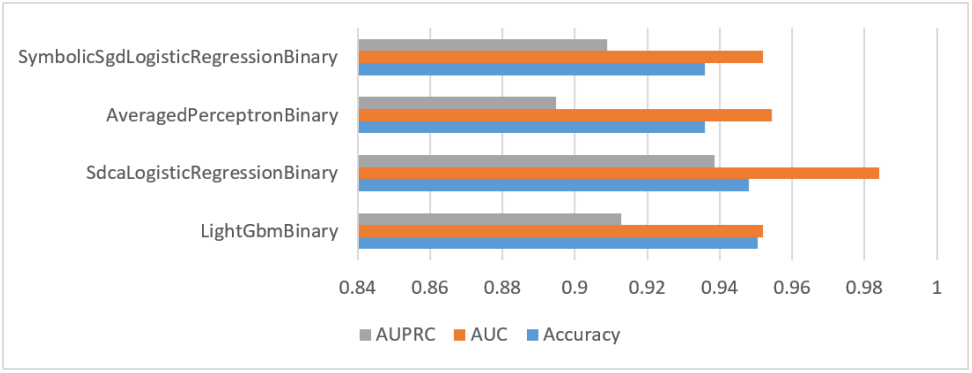


Fig. 4. Results for 60 seconds run

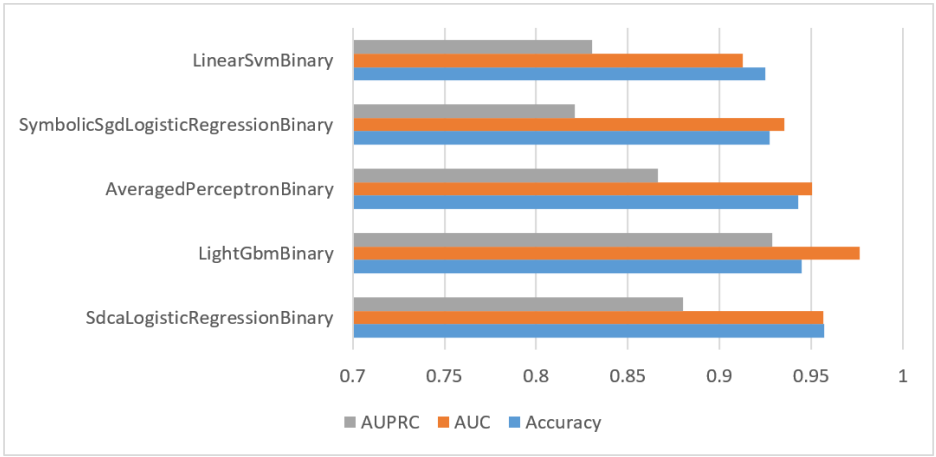


Fig. 5. Results for 180 seconds run

3. DEPLOYMENT AND RESULTS

The model has been included in a "bot" system, which is using a real user account on a system and is posting a comment to a post where the misformatted code has been found – usually because the user forgot to mark the code snippet properly, but just copying and pasting it into the text.

The first test deployment started in October 2019, and the system was limited only to report the found unmarked code problems to the logging system, no actual messages to users were sent. The problems reported were then manually checked and the algorithm was modified to post a comment only if a confidence score for a classification was greater than 0.99, to limit the number of false positives, as the author decided it would be better to not nag the user (as other, real human users can do this) than to nag the user unnecessarily.

The final deployment started in early 2020. The system was prepared as a service completely separate from the discussion forum itself, running on a separate machine and interacting with API only. The overall algorithm for deployed software is presented in figure 6.

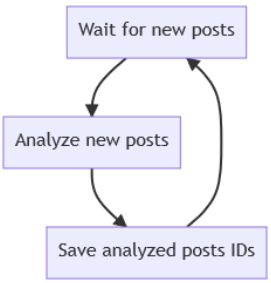


Fig. 6. Overall system architecture

The software is waiting for new posts for a defined time limit, which was set to 15 minutes, then it is analyzing new posts, and saves their IDs to the local datastore, so no new forum post is analyzed more than once.

For each new forum post, the software is checking if the post is on a blacklist – currently blacklisted are some of the subforums, the system should be checking only the code in "technical" subforums, discussing programming languages, not careers or educational materials. Then, the post content is being acquired from the API, and the HTML content is being removed – when the code is properly marked as a code, it is being encased in HTML5 code element, so properly marked codes will be removed in this step. Here additional checks are being run, e.g., removing some of the HTML anchor elements, which also may be classified as code, but they are not actual code snippets left by the user.

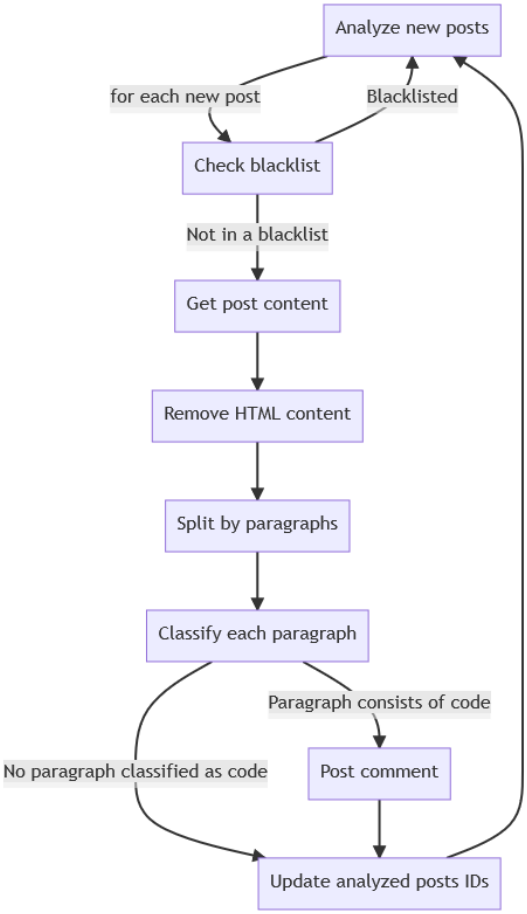


Fig. 7. The analyzer flowchart

Next, the content is being split into paragraphs and each paragraph is being classified by the ML system – if any paragraph of text classification is "code" with a confidence score greater than 0.99 the bot is sending a request to the forum software, creating a new comment to the post saying, "This post may include unformatted code." and link to a help webpage describing the problem.

In the last step, no matter if the unmarked code was found or not, the post ID is saved into the "already analyzed" dataset to prevent double comments.

4. CONCLUSIONS

The authors tried to solve a very specific problem of differentiating between the source code of a programming language and a technical text in the case of very short snippets. Such a problem is not widely discussed, as similar papers are mostly concentrating on differentiating between different programming languages.

The proposed analyzer was built using the AutoML system included in the ML.NET library and then deployed. Until the half of 2021, the deployed solution analyzed over 100,000 posts and published over 500 comments when unmarked code was found. Users of the forum have been informed that the system is in use and were able to report the situation that the system marked the post as unformatted while there was no such case ("false positive") – until the half of 2021, the number of false positives reports was 12, resulting in False Positive Rate of 2.4%.

Tab. 1. Accuracy of differentiating between code and text in case of short code snippet

Library	Accuracy
pygments	45.84%
linguist	1.01%
Proposed solution	94.79%

When comparing the proposed solution with broadly available alternatives, which are suited for a bit more advanced case, the difference in accuracy is astonishing.

To conclude, the system is working properly and living up to the expectations. The AutoML process was easy to use and generated a working model, available to use without much hassle. The author believes that AutoML systems are an interesting case in trying to apply Machine Learning to solve problems where neither the highest possible accuracy nor the knowledge of tuning and applying data models is very important. The particular problem may be exchanged and implemented into a software development process for documentation classification and archiving purposes, but the whole AutoML ecosystem is very useful and may be implemented in the industry in overall.

The software, generated models, datasets, and training logs for the deployed analyzer are available (Badurowicz, 2020) as the Free Software on the GitHub platform.

REFERENCES

4programmers.net. (2000). *Forum dyskusyjne dla programistów*. https://4programmers.net

Ahmed, Z., Amizadeh, S., Bilenko, M., Carr, R., Chin, W.-S., Dekel, Y., Dupre, X., Eksarevskiy, V., Filipi, S., Finley, T., Goswami, A., Hoover, M., Inglis, S., Interlandi, M., Kazmi, N., Krivosheev, G., Luferenko, P., Matantsev, I., Matusevych, S., Moradi, S., Nazirov, G., Ormont, J., Oshri, G., Pagnoni, A., Parmar, J., Roy, P., Siddiqui, M. Z., Weimer, M., Zahirazami, S., and Zhu, Y. (2019). Machine Learning at Microsoft with ML.NET. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 2448–2458). Association for Computing Machinery. https://doi.org/10.1145/3292500.3330667

Alreshedy, K., Dharmaretnam, D., German, D. M., Srinivasan, V., & Gulliver, T. A. (2018). *SCC: Automatic Classification of Code Snippets*. arXiv:1809.07945. https://doi.org/10.48550/arXiv.1809.07945

Badurowicz, M. (2020). *ktos/Eleia: 4programmers.net bot for nagging users when their code in post is not marked as code*. http://github.com/ktos/eleia

Van Dam, J. K., & Zaytsev, V. (2016). Software Language Identification with Natural Language Classifiers. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 624–628). IEEE. https://doi.org/10.1109/SANER.2016.92

Gilda, S. (2017). Source code classification using Neural Networks. *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)* (1–6). IEEE. https://doi.org/10.1109/JCSSE.2017.8025917

GitHub Copilot – Your AI pair programmer. (n.d.). Retrieved January 22, 2021 from https://copilot.github.com

He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, *212*, 106622. https://doi.org/https://doi.org/10.1016/j.knosys.2020.106622

Khasnabish, J. N., Sodhi, M., Deshmukh, J., & Srinivasaraghavan, G. (2014). Detecting Programming Language from Source Code Using Bayesian Learning Techniques. In P. Perner (Ed.), *Machine Learning and Data Mining in Pattern Recognition* (pp. 513–522). Springer International Publishing.

Kłosowski, G., Kulisz, M., Lipski, J., Maj, M., & Bialek, R. (2021). The Use of Transfer Learning with Very Deep Convolutional Neural Network in Quality Management. *European Research Studies Journal*, *XXIV*(Special Issue 2), 253–263. https://doi.org/10.35808/ersj/2222

Kulisz, M., Kujawska, J., Przysucha, B., & Cel, W. (2021). Forecasting Water Quality Index in Groundwater Using Artificial Neural Network. *Energies*, *14*(18), 5875. https://doi.org/10.3390/en14185875

LeClair, A., Eberhart, Z., & McMillan, C. (2018). Adapting Neural Text Classification for Improved Software Categorization. *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (461–472). IEEE. https://doi.org/10.1109/ICSME.2018.00056

Linguist. (n.d.). Retrieved January 22, 2022 from https://github.com/github/linguist

Machrowska, A., Szabelski, J., Karpiński, R., Krakowski, P., Jonak, J., & Jonak, K. (2020). Use of Deep Learning Networks and Statistical Modeling to Predict Changes in Mechanical Parameters of Contaminated Bone Cements. *Materials*, *13*(23), 5419. https://doi.org/10.3390/ma13235419

Madani, N., Guerrouj, L., Di Penta, M., Gueheneuc, Y.-G., & Antoniol, G. (2010). Recognizing Words from Source Code Identifiers Using Speech Recognition Techniques. *2010 14th European Conference on Software Maintenance and Reengineering* (pp. 68–77). IEEE. https://doi.org/10.1109/CSMR.2010.31

Ohashi, H., & Watanobe, Y. (2019). Convolutional Neural Network for Classification of Source Codes. *2019 IEEE 13th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC)* (pp. 194–200). IEEE. https://doi.org/10.1109/MCSoC.2019.00035

Pygments - Python syntax highlighter. (n.d.). Retrieved January 22, 2021 from https://pygments.org

Sobaszek, Ł., Gola, A., & Kozłowski, E. (2020). Predictive Scheduling with Markov Chains and ARIMA Models. *Applied Sciences*, *10*(17), 6121. https://doi.org/10.3390/app10176121

Szabelski, J., Karpiński, R., & Machrowska, A. (2022). Application of an Artificial Neural Network in the Modelling of Heat Curing Effects on the Strength of Adhesive Joints at Elevated Temperature with Imprecise Adhesive Mix Ratios. *Materials*, *15*(3), 721. https://doi.org/10.3390/ma15030721

Ugurel, S., Krovetz, R., & Giles, C. L. (2002). What's the Code? Automatic Classification of Source Code Archives. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 632–638). ACM Digital Library. https://doi.org/10.1145/775047.775141

Wever, M., Tornede, A., Mohr, F., & Hullermeier, E. (2021). AutoML for Multi-Label Classification: Overview and Empirical Evaluation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *43*(09), 3037–3054. https://doi.org/10.1109/TPAMI.2021.3051276

Yin, P., Deng, B., Chen, E., Vasilescu, B., & Neubig, G. (2018). Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow. *International Conference on Mining Software Repositories* (pp. 476–486). ACM Digital Library. https://doi.org/10.1145/3196398.3196408