Preprint of a chapter: Marcel Jerzyk, Jakub Litkowski, Jakub Szańca and Lech Madeyski,
Developments in Information and Knowledge Management for Business Applications,
Studies in Systems, Decision and Control

Evaluating the potential of a candidate for a job offer based on his GitHub profile

Marcel Jerzyk, Jakub Litkowski, Jakub Szańca and Lech Madeyski

Abstract
Context: Nowadays, it is much harder for new programmers to acquire industry experience. This trend started being noticeable at around 2018, but now - due to the coronavirus pandemic - it is even more prominent. Several dozens of junior programmers apply for one job offer and thus there can be a need for a tool that would facilitate the identification of candidates that the employer is interested in. It is considered a good practice for Junior Developers that are seeking to be hired to have public repositories with some of their personal work. The problem is that usually the recruiter does not have the knowledge to use that information, which - in fact - could be very impactful as it is literally the "show-off"" of one's current skills. Right now it is completely disregarded until the later stages of the recruitment phase - after the junior already passes the first recruiter look-up. Only then the repository is examined by a technical interviewer but that also could be a concern for the company as it books out the valuable time of an expertised employee. Moreover, there could be situations when one's CV is completely skipped due to various reasons and thus missing the chance to be employed or have his repositories examined by a professional which could prove otherwise (that one has all the abilities and skills needed for the position).
Objective: This study aims to identify and investigate if its possible to acquire usefull information's from a GitHub profile that could be beneficial for recruiters in a recruitment process. If that would be proven to be an effective way of measuring one's abilities and skills - maybe it could be also used by the developers themselves to attest their own potential quality.
The visible main challenge is the inefficient hiring process with no accurate technical screening methodology. The goal is to improve the rate of recruiting highly skilled programmers by making use of the information's that are already provided in CV's by standard.
Method: We conducted a systematic review using a database search in Scopus and

Marcel Jerzyk, Jakub Litkowski, Jakub Szańca, Lech Madeyski
Wroclaw University of Science and Technology, Poland

evaluated it using the quasi-gold standard procedure to identify relevant studies. We created polls to gather information from developers own experiences and asked them to attach their GitHub profiles to the questionnaire. Then, with machine-learning based approach, we used that data to automatically evaluate the potential of a candidate by giving only the link to *GitHub* profile on input. Information from *GitHub* accounts were scraped by the available *API* and by linting the repositories. In the data sheet used to obtain data from publications, we factor the research questions into finer-grained ones, which are then answered on a per-publication basis. Those are then merged over a set of publications using an automated script to obtain answers to the posed research questions.
Results: Among the classification algorithms that we tested with our model, *SVM* obtained the best results for most of the metrics, regardless of the type and number of selected features. More importantly - the features derived only from the *GitHub API* and by linting repositories, the results between the algorithms did not differ significantly from each other.
Conclusion: We conclude that based on the results, we achieved that there is enough data stored on *GitHub* profiles from which machine learning algorithms could provide a reasonably reliable evaluation of the potential quality of the users behind them as employees, although more research with bigger sample sizes are needed to improve the credibility of these results.

1 Introduction

Junior developers entering the market come across some difficulties with getting hired, especially if they lack practical experience[18].This trend started being noticeable at around 2018[9] but now - due to the coronavirus pandemic - it is even more prominent [13]. While the hiring practices of graduates are not thoroughly studied in the context of the IT industry, the concept in general is well understood Selecting the most valuable candidate for interview is an important process. While the hiring practices of graduates are not thoroughly studied in the context of the IT industry, the concept in general is well understood [16]. As a standard, recruiters look through Curriculum Vitae (*CV*) to categorize all candidates and call them for interview. Nowadays, to minimize the time spent on a single CV, recruiters have various prescreening methods, listing them in order of importance [14]:

- Employment history and previous business-related work
- University and Academic Performance
- Listed programming abilities and other colloquially called *"buzzwords"*
- Soft Skills

The CV could include a lot of important information's, but the recruiter usually spends from 6 to 60 seconds at max per one application [14]. It is surely not enough time to examine candidate skills correctly, more over - the recruiter usually does not even have the industrial knowledge to assess the skills and examine the candidate's quality by browsing through „code portfolio”. It is done only after the prescreening is already done, when the filtered-out list of candidates goes to a professional technical employee that has the required knowledge and skills to examine the potential of a candidate. The information is there - various online hosting websites that allow to showcase one's code are widely used and added to the CV's - but a qualified candidate may be omitted in the prescreening method because his cv-building skills were not good enough to pass.

One of such hosting websites is GitHub [1] - it is a service supporting version control using git, which is used by programmers and developers for storing and managing their project codebases and repositories. The popularity of GitHub in 2021 is well established and still rising. Already in 2020 the website gathered over 36 million users [5] and today, as of March 2021 - there are 55 million users [6]. Dabbish et al. [3] showed that software developers in a public project on GitHub consciously manage their online reputation. They do recognise that others developers can assess their personal characteristics such as the quality of work or commitment by looking, for example, through their commits and judging the quality, checking the amount of forks the project has or the star counter. Singer et al. [17] found that the number of followers one may have on GitHub is also an valuable information.

We recognise that the CV evaluation procedure has several shortcomings and not much has changed since 2013 [2]. These shortcomings especially apply for those who do not have any relevant work experience or formal degree. We also note that GitHub is a place storing lots of very valuable information that could (or maybe should) be used to evaluate the potential of a candidate. Various research works have confirmed that it is possible to retrieve data like technical technical role [12], identifying experts in Software Libraries and frameworks [10]. Because the recruiters have very limited time available to review a single CV, we search for an machine-learning based solution that could help extract information from GitHub by a non-expertised recruiter. This could not only reduce the number of false-positive skips but also increase the overall quality of the list of candidates after the prescreening is done.

2 Systematic Review

What distinguishes this paper among others — as we mentioned — we do recognize that the majority of recruiters do not have skills that could examine the technicality of a candidate. For example in the *"Mining the Technical Roles of GitHub Users"* the survey was conducted among recruiters from StackOverflow website [2] - which is primarily used by developers for the purposes of collaboration & knowledge sharing. Thus, it can be concluded that the surveyed sample was exceptional among the global population of recruiters as they have made extra efforts to create an account on IT-technical website and not only that - they were actively browsing it and even possibly

[1] GitHub Website URL: `https://github.com/`

[2] StackOverflow Website URL: `https://stackoverflow.com/`

creating extra content which made them visible and reachable by researchers. Based on the research we have conducted, there are only a few unfinished recommendation systems that use the user GitHub profile, but at the same time there is some research that focuses on mining data from the repositories.

2.1 Research Questions

Research questions presented below should explain to help understand the importance of creating such a tool and provide information about the functionalities which the potential end user might need (i.e. - recruiters, developers themselves). Furthermore, we also explore issues related to the recruitment itself, as well as the quality of repositories.

Hence, we defined four research questions.

- *RQ1: What are the most common skills the company is looking for among the candidates?*
  Our system needs to determine whether the developer is worthy of attention and if he should be considered further in the recruitment process. The answer to this question should explain what skills are considered important among companies.
- *RQ2: How a given repository makes a good impression?*
  One way of assessing developer expertise is to analyse his projects. Thus, we should determine which features of repositories are the most representative ones.
- *RQ3: Is it possible to get a good enough information about the candidate's potential based on the GitHub profile?*
  This question should provide us with information whether the recommendation system based on repositories is worth devoting time.
- *RQ4: Is GitHub repository used only for programming purposes?*
  It often happens that an item is used contrary to the original intention of the author. The same situation might occur with the repositories. There is no compulsion to prohibit the use of a repository for purposes other than development.

2.2 Currently Available Resources on the Subject

We used search engines (Google Scholar and Scopus) to find proper resources and currently available papers on the subject that we are investigating in our paper, and then - to collect all relevant literature - we created a more specific search string which is given below *[ref: 1]*.

Listing 1 Search String Query

```
(
  skill AND
  repository AND
  ( github OR gitlab OR bitbucket )
) OR (
  recruitment OR hiring OR interview
) AND (
  programming AND purpose
) AND (
      "Computer␣Science"
) AND (
  mining AND software
)
```

Because of the enormous number of results in Scopus, we decided to add another limitation such as the required presence of „Software Engineering” keyword and „2017-2021” year constraint to narrow it down in time to only the most relevant studies.

This gave us a good amount of publications which needed to be further filtered by hand. As shown on the literature identification figure *[ref: 1]* search engines provided us with over two hundred publications. Unfortunately, many of them did not answer the previously established research questions and many of them were simply not in English nor Polish language so that we can understand them. After all, we have identified ten of the most important studies that are helpful in our topic.

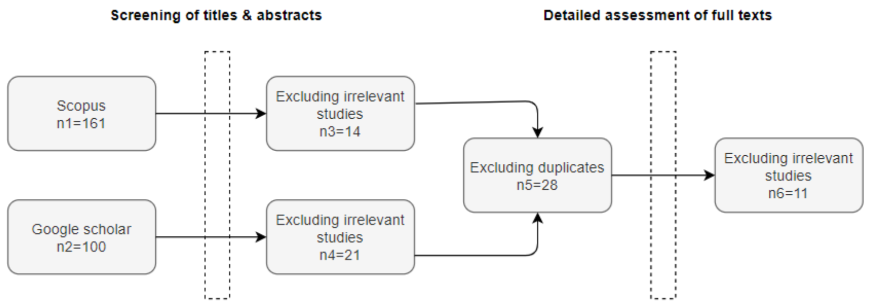

Screening of titles & abstracts
Detailed assessment of full texts
Scopus n1=161
Google scholar n2=100
Excluding irrelevant studies n3=14
Excluding irrelevant studies n4=21
Excluding duplicates n5=28
Excluding irrelevant studies n6=11

Fig. 1 Literature Identification

2.3 Results Selection Process

After running the search string query, we were presented with a large number of results. We have gone through the results list and discarded all duplicate entries and simultaneously discarded those papers whose titles or abstracts were not related to the topic of our research. Thus, we ended up with 28 papers which we further examined. Finally, we have chosen 10 papers which are closely related to the subject of this paper.

3 Literature Review

We can divide the most important skills possessed by developers which are in demand by the hiring companies into two categories: *Soft Skills* and *Hard Skills*. Most Both of these skill groups are equally valuable in the eyes of companies and the most important ones are going as follows [11]:

- **Hard Skills:** Languages, Libraries & Frameworks, OS & Infrastructure, Process & Methods, Data Systems, and Development Tools
- **Soft Skills:** Communication, Collaboration, Problem-Solving

These findings were based on job offers which were listed on *Stack Overflow* website. That should be emphasized because this website is used primarily by developers for problem-solving and discussion - the job listing functionality is there as a side feature and that could lead possibly to results that are more technical.

Still, one of the most important factors is collaboration [20] and even though it is indeed possible to dig out and assess technical skills as we have mentioned earlier, it was shown that there is a lack of any strong association between them and social skills [20]. Although it could be complicated to obtain information about non-technical skills, there is possible to dig out this information's via GitHub Profile in conjunction with CV and Linked In[3] profile with satisfactory results [4]. This is a small disclaimer but if ever it would be necessary to verify that the attached GitHub profile belongs to the applicant for whatever reason it is possible [7] mainly via `README.md` natural language analysation.

Surely, companies are also interested in the expertise of the potential candidates in a given set of languages and sometimes even precisely - frameworks. This is essential because hiring a developer in a senior position without real seniority is a very costly mistake given the current salaries for any senior position. There is also a lot of potential here, as it was shown that such information can be fairly successfully extracted with over 70% (on average) success rate, particularly for distinguishing React, Node-Mongodb, and SOCKET.IO experts [10].

[3] Linked In is an employment-oriented online service that operates via websites and mobile apps. The platform is mainly used for professional networking, and allows job seekers to post their CV's and employers to post jobs.

One would ask whether such a tool could work in practice as maybe there are too many completely irrelevant repositories that are not project-based. Luckily, this question was already addressed in the past and it turns out that not every repository is used just for programming purposes. Many of the repositories are abandoned after a few weeks - there is a substantial spike in activity in the first few weeks, but as time passes by, the activity is decreasing drastically (logarithmically to be exact). A good number of people are using repositories just as a storage place and there is also a noticeable amount of empty repositories as an effect of some experiments or learning attempts.

In fact, a quite similar tool was already developed [19] but it used *GitLab* instead of *GitHub* to aggregate user data which gives u a green light to pursue with the project. They managed analysed six conditions: *code quality, code quantity, skills, contributions, personalized commit time, and project participation* and then used them to evaluate developer expertise. It was concluded that none of these could be chosen as the most significant and their importance was about equal. That is why our project is quite relevant as we address the similar, already detected issue, but we want to pursue with a much more popular service to hit bigger „audience" of candidates [15].

We also recognize that there are few pitfalls that could lead to false-positive results, for example, if we take a deeper look into the on-boarding programs. Briefly, on-boarding programs rely on mentored work with open source contributions. It was demonstrated that this practice is not as effective at transitioning new developers into long-term contributors as it was previously speculated, although developers who after all do succeed through these programs are valuable. [8] That's why we are aware that not every contributor to the open-source is going to be a skilled programmer, even though he could receive many tips after being mentored by a professional, and moreover - he would most likely contribute with high-quality code which was not purely developed by the person behind the GitHub profile.

Nevertheless, such a tool is really in demand. The current recruitment process causes a lot of stress and is not liked among developers. Programmers say that during the interview they can not show their whole potential and they are served with exercises that are not useful during normal work and do not check their actual knowledge. Developers criticize big companies for their ease of rejecting candidates just because there are just so many more available candidates and even if they reject someone who is qualified, it will not really hurt the company because they will find somebody else. [1]

4 Methodology

In this section, we defined the methods, techniques, tools, and processes referring to the main part of our research.

4.1 Data Collection

4.1.1 Questionnaire

Our main goal during the data collection process was to obtain GitHub Usernames along with as many answers to questions related to the subject of the recruitment process, an assessment of own skills as well as some additional information which could help in the reconstruction of results found in other papers. We have created a Google Form questionnaire and then posted it on two programming-related Facebook Groups.

In both groups, one can find people with a full range of professional experience but based on the activity (posts and comments), one of them is mainly characterized by highly qualified employees (Seniors) while the other seems evenly distributed.

4.1.2 Parsing General GitHub Information

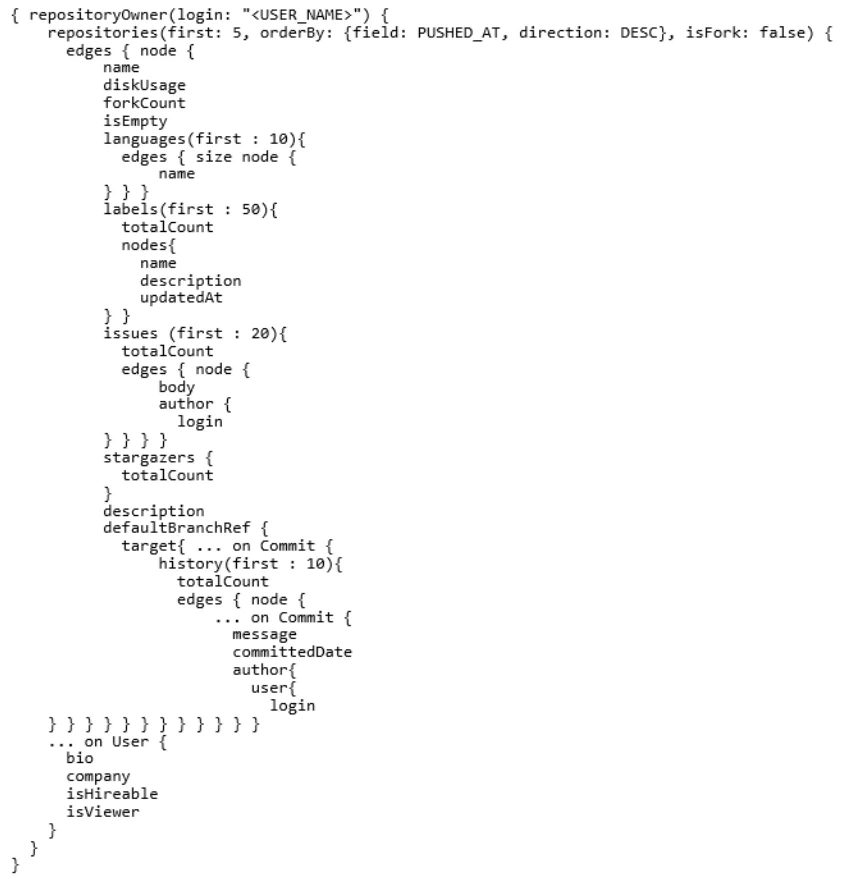
To collect data from GitHub, we used GraphQL Queries [4]. The query takes only the username on the input and on the output puts only the information precisely specified within the query itself. GraphQL Query is presented in the image below *[ref: 2]*.

Requests were done within R script for every username, which was extracted from a GitHub link that was provided in the questionnaire. Unfortunately, not every participant provided a relevant link to the repository or even did not provide any. First, the script loads the questionnaire data and then for every username executes the request to *GitHub API*. In case of an invalid link, it returns nothing. After collecting data for every user who took part in the questionnaire, the script combines the survey data with the information retrieved via *GitHub API* and saves it as `csv` file.

4.1.3 Parsing GitHub Repositories

This part requires quite a bit of time to complete per just one single user. The time it takes to fetch and parse is mainly related to the number of repositories that a given user has as well as the contents which are in them. The script receives on input one single username and then it collects all repository names into a list. In the next step, all of these repositories are fetched from remote to the local computer so they can be

[4] GraphQL GitHub API: `https://docs.github.com/en/graphql`



```
{ repositoryOwner(login: "<USER_NAME>") {
    repositories(first: 5, orderBy: {field: PUSHED_AT, direction: DESC}, isFork: false) {
      edges { node {
          name
          diskUsage
          forkCount
          isEmpty
          languages(first : 10){
            edges { size node {
                name
          } } }
          labels(first : 50){
            totalCount
            nodes{
              name
              description
              updatedAt
          } }
          issues (first : 20){
            totalCount
            edges { node {
                body
                author {
                  login
          } } } }
          stargazers {
            totalCount
          }
          description
          defaultBranchRef {
            target{ ... on Commit {
                history(first : 10){
                  totalCount
                  edges { node {
                      ... on Commit {
                        message
                        committedDate
                        author{
                          user{
                            login
    } } } } } } } } } } } } }
    ... on User {
      bio
      company
      isHireable
      isViewer
    }
  }
}
```

Fig. 2 GraphQL Query

checked via *Mega Linter*. After collecting all of the repositories, *Mega Linter* task is executed in each single one of them. When that is done, the script moves on to the merge task, which transforms all the generated `jsons` into one, single `json` file, so that it can be easily imported and accessed by the model in *R*.

4.2 Data Preprocessing

4.2.1 Questionnaire

In total, we received 67 completed questionnaires and 45 of them were provided with GitHub profile. The data file was further cleaned and reformatted with the use of

Python [5] script. Both the original and formatted `csv` files can be found in the project source under the name „`questionnaire.csv`" and „`cleaned_data.csv`" as well as the Python script used for data formatting which can be found in „`py_scripts`" directory under the name „`questionnaire_adjuster.py`". To run the script, one must have Python 3.9 or newer installed on the machine along with „Numpy" [6], „Pandas" [7] and „Requests" [8] packages.

Delving into the details, starting with the most important - 7 answers were manually modified to standardize the values into one datatype (`int`), as they were given in the questionnaire as strings, so that people are not constraint to the few selected numerical values and - unfortunately - there was no other way to have an input field that allows only for `UnlimitedNatural` numbers. Majority of these answers were given as ranges (ex.: 10 – 20, *a dozen*, 30 – 40) - these were modified to be the average of the given range. The "self-doubtable" and the „not exactly sure" answers like 60?, 4/5 and 20+, were changed to the lowest value in the suggested range.

Besides that - all available checkbox answers (*Soft Skills* & *Pre-work Experience*) were transformed so that each of them has their own respective column with `Boolean` answers whether it was checked or not.

Other changes are technical and linguistic so that the data set could be reused with comprehension without the knowledge of Polish language:

- Unicode Characters (like *Emojis*) were removed from the column names, and the column names themselves are now English words
- Every single column had their data properly adjusted so they could be interpreted with their suitable data type
- Similar columns from different sections of the questionnaire which were the result of answers to similar but rephrased questions merged
- Answers which were given as strings like „`3,5 year`" were transformed into `integers` in units of months

4.2.2 Repositories

One of the most difficult tasks is to properly obtain information from the repositories of a given user. For research purposes, we decided to track them for potential errors and warnings which can be identified via linters. Linter is a static code analysis tool used to flag any bugs, errors, stylistic warnings, suspicious constructs, redundant code, and more depending on the language and/or tool. Of course, the user could have repositories with code written in any language that exists, and that is a real problem, which we mitigated by a linter-aggregating tool - *Mega Linter* [9]. Mega Linter is an open-source tool that simply detects the languages used in a given

[5] Python Website: `https://www.python.org/`

[6] Numpy PyPI Page - `https://pypi.org/project/numpy/`

[7] Pandas PyPI Page - `https://pypi.org/project/pandas/`

[8] Requests PyPI Page - `https://pypi.org/project/requests/`

[9] Mega Linter GitHub Page - `https://github.com/nvuillam/mega-linter`

project and then uses all available linters to scan it through. After the scan, it prints out a summary table with the number of Files that were detected and scanned with a given linter, the number of fixed files automatically during the run time, and the number of errors that couldn't be automatically fixed. There's also a second table that's printed somewhere in the first half of the output log that has information about detected duplicate lines and tokens in the project. To obtain that data we redirected the output stream into a file and then parsed it with Python script „`scrape.py`” which can be found in *„py_scripts”* folder.

The usage of the Mega Linter and scrape scripts is more widely described in the main `README.md` file although, in short, one must have Docker [10] and Python installed. Then, simply navigate to the repository which you would like to lint and run a command *[ref: 2]* which will generate an `output.txt` file.

Listing 2 "Running *Mega Linter*"

```
npx mega-linter-runner --flavor all
                        -e 'ENABLE=,DOCKERFILE,MARKDOWN,YAML'
                        -e 'SHOW_ELAPSED_TIME=true'
                        > output.txt
```

Copy the file and paste it in the *scrape.py* directory. Finally use shell scrape command *[ref: 3]* command which will generate an `output.json` with a list of dictionaries structured as in output example given below *[ref: 4]*.

Listing 3 "Launching `scraper.py`"

```
python scraper.py -f output.txt
```

Listing 4 "Parsed Linter Output in `.json` format"

```
{
 "language": str,      # detected language
 "linter": str,        # checked via linter (name)
 "files": int | str,   # detected files in given language
 "fixed": int,         # fixed errors automatically
 "errors": int         # errors that could not be fixed
},

# or

{
 "language": str,      # detected language
 "files": int,         # in a given language
 "lines": int,         # in a given language
 "tokens": int,        # ("chars") in a given language
 "clones": int,
 "duplicate_lines_num": int,
 "duplicate_lines_percent": float,
 "duplicate_tokens_num": int,
 "duplicate_tokens_percent": float
},
```

[10] Docker Website - https://www.docker.com/

4.3 Creating a Machine Learning Model

We decided to chose `mlr` [11] package as a tool which assists in creation of machine learning models and which is by far one of the most convenient machine learning packages in R [12]. We have performed the following processes:

As our first step, we began by importing the data from the `csv` file [13] with appropriate `UTF-8` encoding into a data frame. Then, in our first approach to the task, we have split the data into two separate data sets, one of which is the training set which consist of 75% of our data and the other - testing set with the remaining rest.

Now, after the data set is ready, we can proceed towards the creation of the machine learning model. First, we have defined a learning task for classification. We have specified `Task ID`, `Process Data` and `Target Column` (target column holds the information whether a given user is interesting in terms of his potential as an employee - or in other words - „attention worthy").

We have constructed a *learner* by calling `makeLearner()` method with *Random Forest* classification algorithm and afterwards, we proceed to `train()`. That gave us the predicted target values for our data set. Then we started to `predict()`. We also called `makeResampleDesc()` method with which we determined the resampling strategy. To estimate precisely how accurately our predictive model performed in practice we have chosen *cross-validation*, more specifically - *10-fold cross-validation*.

This was the time to begin the most rewarding part of the study. Having collected the data from all previously mentioned sources (which is the survey among programmers and IT specialists *[ref: 4.1.1, 4.2.1]*, the mega linter scans of repositories of the surveyed developers *[ref: 4.1.3]* and the various information available through the GitHub API *[ref: 4.1.2)]* into one single `csv` file, we began the work.

First, we filtered out those users which did not provide their GitHub username and those who did not classify any „main" language (and two more users whose repositories could not get linted because of the limited available time to perform such operations or one more user who had no public repositories on his GitHub profile). The remaining pool of 26 was the base from which we extrapolated the features that were used in the machine learning model:

- `ExperienceDuration` - The time that a person is being present on the labor market.
- `WorkFindTime` - The time that elapsed until the first job by a given person was found.
- `AvgCommitTime` - The average time that between commits. Missing values were filled by inserting the averages.
- `ExpType` - Weighted average of the type of experience the person has.

[11] `mlr` Website: `https://mlr.mlr-org.com/`
[12] R Website: `https://www.r-project.org/`
[13] The `csv` file is available at the „`./data/cleaned_data.csv`" path in repository of this paper

- `SoftSkills` - Weighted average of the soft skills that a person had listed on their CV, in which the most important factor of 2 was given to the *communicativeness*, while the rest is set as 1.
- `DupLinesPercent` - The percent of lines that were detected as duplicated throughout a given repository summed over all repositories.
- `LinesPerFile` - The ratio that is calculated via dividing of the number of lines by the number of files.

The data was labeled on the basis of the current analysis of available information about the users. Then, we proceed to analyse the correlations between variables in that data.

In our research, we decided to use three - fairly common - classifiers:

- `K-NN` - Method of the *K Nearest Neighbors* focuses on the search for objects which are closest to the object of classification, and then it tries to determine the class of new objects based on its previously found neighbours. `K-NN` is resistant to the insulated values.
- `Random Forest` - The main idea behind the algorithm is metaphorized by a forest which is not a coincidence - it creates lots of decision trees based on random data. Decision-making, i.e., classification, is performed as a result of *majority voting* over the classes indicated by specific decision trees. This method maintains accuracy even if some data values are missing. It also is resistant to over fitting.
- `SVM (Support Vector Machine)` - The concept behind the *SVM Classifier* is the division of space using a function which splits the space into two areas, which correspond to two decision classes. This method works well with a small teaching sample as well as with a large number of attributes.

We tested the effectiveness of the models using 5 indicators which assess the quality of prediction:

- `MMCE (Mean miss classification error)` - it is the average ratio of miss classified classes calculated as $\frac{f_p+f_n}{t}$, where: f_p is *false positives*, f_n - *false negatives* and t is denoted as *total*.
- `MCC (Matthews correlation coefficient)` - it is an indicator of the quality of the binary classification. It is a measure of the correlation between true and predicted values. The *MCC* takes into account all four values from the confusion matrix - when $f_p == f_n == 0$ then the *MCC* value is 1, and when $t_p == t_n == 0$, the *MCC* value is -1.
- `ACC (Accuracy)` - one of the core and the most commonly used indicator; it tells how often the classification is correct. It is calculated as $\frac{t_p+t_n}{t}$. This measure is sensitive to unbalanced data sets.
- `F1` - harmonic mean of Precision indices (ratio of t_p to all objects classified as `true`) and Recall (ratio of t_p to all observations in a given class ($t_p + f_n$)).
- `Kappa` - the reliability coefficient of duplicate measurements of the same variable. The closer it is to the value of 1, the more consistent the ratings are. In turn, the closer to the value of 0, the more divergent the assessments are.

4.4 Results

We decided to compare the results for three different models. Below you can find two tables - one of them is based on the results which were achieved using all the available data sources we have (which includes the questionnaire) *[ref: table1]* while the other uses only the data that can be gathered using GraphQL request to GitHub API and the Mega Linter with parsing scripts *[ref: table2]*. In the second case, we used `AvgCommitTime`, `DupLinesPercent` and `LinesPerFile` attributes. Below the tables, we present the correlation matrix of individual features *[ref: figure3]*.

Algorithm	**MMCE**	**MCC**	**F1**	**ACC**	**Kappa**
Random Forest	0.329	0.245	0.410	0.671	0.219
SVM	0.315	0.356	0.422	0.685	0.311
KNN	0.388	0.224	0.431	0.611	0.184

Table 1 Measures scores for each classifiers (all features)

Algorithm	**MMCE**	**MCC**	**F1**	**ACC**	**Kappa**
Random Forest	0.360	0.299	0.451	0.639	0.245
SVM	0.319	0.314	0.451	0.680	0.280
KNN	0.367	0.237	0.484	0.632	0.209

Table 2 Measures scores for each classifiers (*GitHub API* and *Mega Linter* features)

5 Discussion

5.1 All Features

The variables prepared for the machine learning model were not correlated with each other. No substantial correlation was observed between the percentage of duplicate lines and the time until the first job was found. Unnoteworthy although reassuring connection, which has the highest correlation among those all variables, between the experience duration and *WorkFindTime* is present.

Taking into account the results of the most common classifiers (*ACC* and *MMCE*) we conclude that the *Random Forest* and *SVM* algorithms coped best with the classification task. The *SVM* had the best results, which is not a surprise - as in theory - it should produce solid results even with a small amount of training data. That also reassures the validity of this algorithm as it is „field-tested” in other research papers with similar data size with success. *KNN* which by definition is the

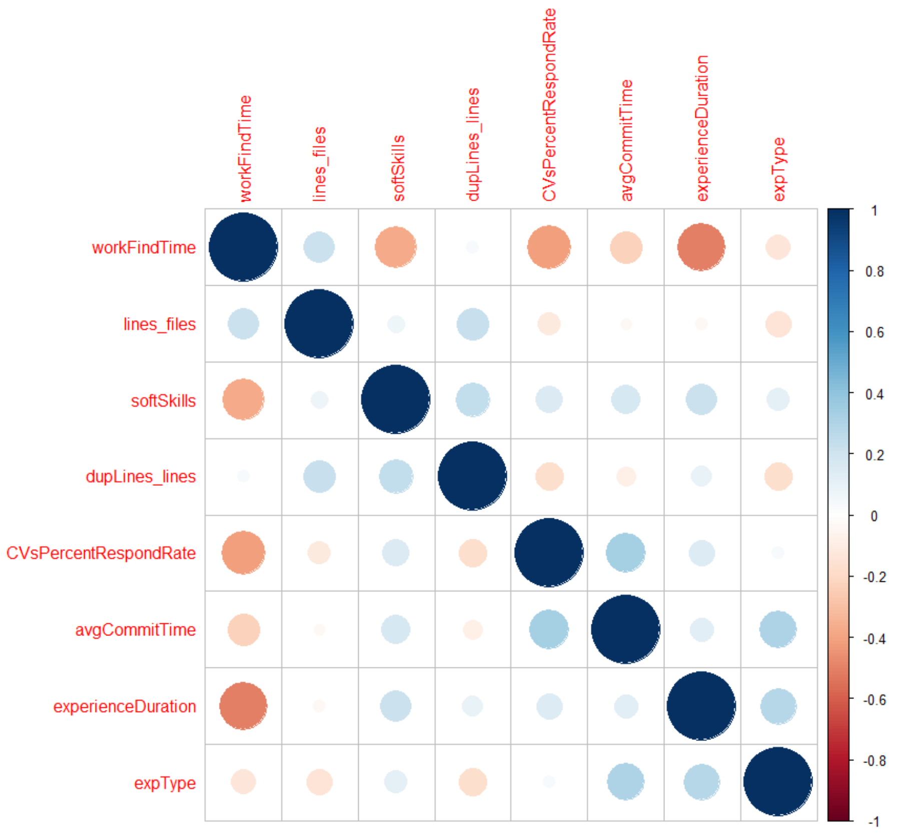


Fig. 3 Correlation Matrix

simplest, performs the worst because it requires features in a uniform scale to classify with its best efficiency while our data has features of quite different scales.

However, one should have doubts about the quality of these indicators (the *ACC* and *MCEE*) because the data that we feed into the algorithm is not perfectly balanced (the % share of classes in the set is about 62% to 38%). The *F1* measure assessed the quality of all models on comparable level. On the other hand, the *MCC*, which is often considered a key indicator, showed that the *SVM* model was by far the best in classifying tasks.

The *Kappa* coefficient for the *Random Forest* and *SVM* algorithms is in the range of 0.21 – 0.40, therefore the *Level of Agreement* can be describe as *„fair agreement”* (which is about an average result). For *KNN* the compliance can be defined as *„slight agreement”*.

To sum up, the most effective model during the classification was the one using the *SVM* algorithm. It was the best according to almost all indicators.

5.2 GitHub Data Only

Comparing the results of the classification using different features, we observed that a different number of features influenced each model in a different way. Each of the measures in the model that used *SVM* algorithm, has worse value (besides the *F1* indicator, which is better for each algorithm). This indicates that the algorithm performs better when it has access to the higher variety of features.

The *KNN* algorithm performance improved as the number of features decreased. This may be due to the fact - as we previously mentioned - that it should perform best with the features of similar scale. As for *Random Forest* - lower number of features made the *ACC* indicator significantly worse while improving *MCC*. It is difficult to explain the reason why such a difference in the results was obtained in this particular case.

5.3 Run-down

Summarizing, in most cases, the results of classification based on fewer features are worse, but the difference is not dramatic. Therefore, it can be concluded that the data obtained from the repository using our method of combining *GitHub API* along with *Mega Linter* tool and our parse scripts is to some extent sufficient to assess the potential of the GitHub user in terms of his potential for an employee.

6 Conclusions

6.1 Further Research

Collecting relevant data is difficult due to developers high understanding and concerns regarding privacy and data collection. Regardless, to improve the credibility of our analysis, one would have to gather larger sample data - we provide working tools and solutions which can help scrape the information and quickly start investigations, just data gathering is a difficult task in of it self.

Easier and more instantaneous way to improve on this paper would be by reproducing what we have done - the logic, tools and parsed data is already here and we believe much better results can be achieved by tweaking the parameters in the algorithms or for example, by using different classifications. Not only that - there is also a possibility to change the information that labels *errors*/*fixed* provide by using the *linters* with different parameters.

6.2 Summary

Our research is a solid brick in the foundation of automatic skill assessment based on the data that is already shared by and available to use on version control websites. By no means - the task is not easy (which was also already concluded by similar research that focused on *GitLab* rather than *GitHub*) and we confirm on independently gathered data with our own methods and algorithms that the selection of criteria is still quite enigmatic and is not clear out.

One of the main concerns could be that not every single user has some meaningful repositories available publicly on their *GitHub* profile, but our analysis has disproved this as the majority of developers do have some kind of viewable personal projects on their *GH*. Even more - majority of developers in the context of our research situation - that is - job search, should have some kind of work available to look up.

In that scenario, we conclude that the GitHub profile is able to provide a sufficient amount of data to assess the potential quality of a developer with reasonably reliable evaluation results.

References

1. Behroozi, Mahnaz, Parnin, Chris, Barik, and Titus. Hiring is broken: What do developers say about technical interviews? In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–9, 2019.
2. A. Capiluppi, A. Serebrenik, and L. Singer. Assessing technical candidates on the social web. *IEEE Software*, 30(1):45–51, 2013.
3. L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work", 2012.
4. R.G.U.S. Gajanayake, M.H.M. Hiras, P.I.N. Gunathunga, E.G. Janith Supun, Anuradha Karunasenna, and Pradeepa Bandara. Candidate selection for the interview using github profile and user analysis for the position of software engineer. In *2020 2nd International Conference on Advancements in Computing (ICAC)*, volume 1, pages 168–173, 2020.
5. GitHub. (Archive) GitHub User Search Query in Janaury 2020, 2020.
Url (accessed: 10.04.2021):
`https://web.archive.org/web/20191016232523/https://github.com/search?q=type:user&type=Users`.
6. GitHub. GitHub User Search Query in March 2021, 2021.
Url (accessed: 10.04.2021):
`https://github.com/search?q=type:user&type=Users`.
7. Hauff, Claudia, Gousios, and Georgios. Matching github developer profiles to job advertisements. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 362–366, 2015.
8. Adriaan Labuschagne and Reid Holmes. Do onboarding programs work?, 05 2015.
9. Melissa Mcewen. Who Killed The Junior Developer? Technical report, Medium, 2018.
Url (accessed: 31.05.2021):
`https://melissamcewen.medium.com/who-killed-the-junior-developer-33e9da2dc58c`.
10. J. E. Montandon, L. Lourdes Silva, and M. T. Valente. Identifying experts in software libraries and frameworks among github users. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 276–287, 2019.

11. João Montandon, Cristiano Politowski, Luciana Silva, Marco Valente, Fabio Petrillo, and Yann-Gaël Guéhéneuc. What skills do it companies look for in new developers? a study with stack overflow jobs, 11 2020.
12. João Eduardo Montandon, Marco Tulio Valente, and Luciana L. Silva. Mining the technical roles of github users. *Information and Software Technology*, 131:106485, 2021.
13. Mar Mustafa. The Effects of COVID-19 on Junior Developers-2020 Job Market. Technical report, Medium, 2020.
    Url (accessed: 31.05.2021):
    `https://amustaf.medium.com/the-effects-of-covid-19-on-junior-developers-2020-job-market-8027b42f0572`.
14. Katarzyna Sławińska Oleszek. To include or not to include? Resume writing webinar by career services of WUST at Santander Scholarship Top Skills Language Academy PWr, SJO, 2021.
15. Thomas Peham. GitLab vs GitHub: Key differences & similarities. Technical report, Usersnap, 2020.
    Url (accessed: 31.05.2021):
    `https://usersnap.com/blog/gitlab-github/`.
16. Kjetil Raaen and P. Lauvås. How companies find and evaluate graduate computer programmers. In *NIK*, 2018.
17. Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Treude, Margaret-Anne Storey, and Kurt Schneider. "Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators". Technical report, University of Victoria, June 2012.
18. Austin Tindle. The Plight of the Junior Software Developer. Technical report, Medium, 2019.
    Url (accessed: 31.05.2021):
    `https://medium.com/swlh/the-plight-of-the-junior-software-developer-494ff1fc4392`.
19. Jing Wang, Xiangxin Meng, Huimin Wang, and Hailong Sun. An online developer profiling tool based on analysis of gitlab repositories. In Yuqing Sun, Tun Lu, Zhengtao Yu, Hongfei Fan, and Liping Gao, editors, *Computer Supported Cooperative Work and Social Computing*, pages 408–417, Singapore, 2019. Springer Singapore.
20. Cheng Zhou, Sandeep Kaur Kuttal, and Iftekhar Ahmed. What makes a good developer? an empirical study of developers' technical and social competencies. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 319–321, 2018.