Job Title Prediction of Github Users

Tuğrul Hasan Karabulut
Boğaziçi University,
tugrul.karabulut@boun.edu.tr

Abstract

Github is a platform, based on Git, that lets us host our code repositories. Aside from this basic definition, it is a showcase for millions of developers to make their portfolios to be seen, contribute to some of millions of available open source projects and interact with other people. In this study, we work on designing a machine learning pipeline to predict the job titles of Github users.

1. Introduction

Github serves several purposes for its users. In its most basic usage, we can upload our code repositories on Github to create a portfolio for our job applications. Also, almost all of the popular open source libraries, frameworks, etc. are maintained on Github. We can read or even contribute to the work done on these open source projects.

Github is also a huge social network like Facebook, Instagram, Twitter etc. Users can interact with other users by following them, starring (similar to like) their projects and communicate with them via *issues*, *pull requests*, etc.

We can leverage this rich information on Github to create useful machine learning models. In this study, we focus on predicting job titles of users. We approach this subject as a classification problem. Later, this model can be used on analyzing different user profiles on Github, recommending job ads to users (StackOverflow has this feature.) and many other work.

2. Previous Work

This work is primarily inspired by MUSAE (Rozemberczki et al., 2019), in which the authors work on different algorithms that produce node embeddings on graphs. One of the datasets they have produced for benchmarking is Github Social Network, which is also present in Stanford Network Analysis Project (SNAP) (Leskovec & Krevl, 2014). It is a labeled dataset that can be used for classification. It is a graph dataset that represents users as nodes and relationship between users as edges. Each node has features related to the user. Features are not clearly specified in the paper. And as labels, it contains two classes: Web Developer and Machine Learning Developer.

3. Method

This work consists of different parts. In this section, we give the details of the study in a step-by-step manner.

3.1. Data Source

Github Social Network is used as the reference dataset. It contains 37,700 users. This user set is used throughout the study. Users who are no longer available on Github are removed. Using this user set, new set of features are extracted to be able to interpret the results and do data analysis more easily.

Github Rest (API) allows us to fetch various information about users and repositories in Github. This API is used to get information about users, their repositories and their follower information.

Follower & following information is re-fetched from Github Rest API to have the current data. For every user, username of the people that user follows or being followed by that user are received from the API. Using this information, a new graph is built with the current information.

Programming languages and keyword information of every user's last 5 repository are extracted. For every repository, a histogram of programming languages are calculated. For example: a repo that contains Python and Java code in equal size in terms of bytes has an histogram with 2 bars of equal size for Python and Java. For keywords, a list of keywords for every repository are received. These keywords are specified by the author of the repository. For example, a user who created a repository for a work on deep learning algorithms for a natural language processing problem can input keywords like ”deep-learning”, ”natural-language-processing”.

Finally, repository-based information is aggregated to have one dictionary of information for every user. Programming language histograms are summed and normalized. That way, we had values for each user that specify, in what percentage do they use X programming language in their work. Keyword information is similarly summed to have an aggregate information of interest areas of the users. Therefore, we can view information like how many ”deep-learning” repositories a user have and so on.

Note that the reason for only using information from last 5 repository is time limitations. Github Rest API has a specific request limit for 1 hour period. So, fetching the information mentioned above takes 1-2 days.

3.2. Feature Extraction

There are many programming languages and infinitely many keywords. Creating a vector similar to one-hot encoding vector will result in a large vector of size $|\{\text{All Programming Languages}\}| + |\{\text{All Keywords}\}|$. To avoid memory issues and curse of dimensionality problem that can occur during modeling, most common 50 programming languages and most common 150 keywords are used.

Additionally, company information is used to extract user-based features. Because company information is created by users using a free text field, we needed to do some text analysis. As a simple solution, most common 100 n-grams (where n=1,2) are extracted from the dataset and a one-hot vector of size 100 is created for each user.

Finally, 4 aggregate features that are present in Github Rest API are used: number of public repos, number of public gists, number of followers, number of following.

In total, $50 + 150 + 100 + 4 = 304$ features are used throughout the study.

3.3. Feature Selection

Experiments have been done with two different feature selection techniques.

Variance thresholding is applied to remove the features with low variances. After examining the variances of the features, it seemed reasonable to experiment with threshold values between 10^{-3} and 10^{-2}.

Model-based feature selection is another method that is applied for eliminating the unimportant features. It can be applied with models that learn a coefficient vector (SVM, Logistic Regression etc.) or models that can produce feature importance values (Decision Tree). For coefficient-based models, a threshold of 10^{-5} is used. For models that produce feature importances, median threshold is applied. Features with coefficient/importance lower than the predefined threshold are discarded. Two models are used for this method: SVM and Extra Trees Classifier. Extra Trees Classifier model works by creating a forest of randomized decision trees (using random splits) (Geurts et al., 2006). Because it’s a tree-based algorithm, it can produce feature importances using impurity reduction values for each feature. Note that the results may differ for extra trees classifier on different runs because of randomization.

3.4. Labeling

”Bio” field that is provided by the API is used to create labels. 4 different job titles and keywords related to a job title are defined. User bios that match a keyword of a job title are labeled with that job title. Predefined job titles are: ”Web Developer”, ”Mobile Developer”, ”AI/ML/Data Engineer”, ”UI/UX Developer”. For example, a user who typed ”machine learning” in his/her bio is labeled with ”AI/ML/Data Engineer”. Similarly, there are many other keywords for each of the job type. With this method, almost 30% of the users labeled. This amounts to almost 10K of 30K users. Enriching this labeling method is another study on its own.

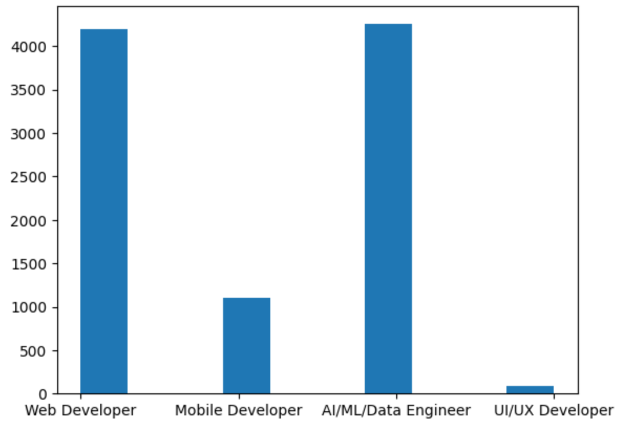

Figure 1. Job title distribution in the dataset.

3.5. Modeling

Two different type of modeling is applied on the dataset. First one is classical machine learning. Second one is basic graph deep learning models that can work on graph data by default.

3.5.1. Classical ML Models

Naive Bayes and Logistic Regression algorithms are applied on the dataset. For Naive Bayes, multinominal version is used. It can both work on discrete and fractional features (MultinomialNB). For logistic regression, multiple logistic regression model is trained in a one-vs-rest manner to create a probability value for each class.

Additionally, to able to do a fair comparison with graph-based models, manual neighborhood features are extracted for these type of models. Specifically, mean of the neighbors (followers and following) are calculated for each user. So, for these models we had $304 + 304 = 608$ features are used.

3.5.2. Graph Deep Learning Models

Two graph models are applied on the dataset.

Graph Convolutional Network is one of the models. It leverages adjacency matrix to do graph convolution operation on the graph (Kipf & Welling, 2016). Architecture used for this study contains only two layers of graph convolution, so we use the information of 2-hop neighborhood.

Model is basically as follows:

$$Z = \text{softmax}(\hat{A}\ \text{ReLU}(\hat{A}XW^{(0)})W^{(1)})$$

where $X \in R^{N\times 304}$ is the feature vector, $W^{(0)} \in R^{304\times d}$ and $W^{(1)} \in R^{d\times 4}$ are learnable weight matrices, d is the hidden dimension size, $\hat{A}$ is the adjacency matrix normalized by node degrees. ReLU and Softmax are the activation of first and second layers, respectively.

GraphSAGE is an inductive graph learning algorithm that works by aggregating features from the local neighborhood (Hamilton et al., 2017). This aggregation part makes it the algorithm an inductive algorithm and therefore it can unseen nodes as well as the nodes present in the graph. Different aggregation techniques can be used on this model. For this study, most simple one, "mean" aggregation is applied.

Here are the feed-forward calculations for node i:

$$Z_i^{(0)} \leftarrow X_i^{(0)}$$

$$Z_{\mathcal{N}(i)}^{(t)} = \text{Mean}(\{Z_j^{(t-1)}, \forall j \in \mathcal{N}(i)\})$$

$$Z_i^{(t)} = \sigma^{(t)}(W^{(t)}[Z_i^{(t-1)}; Z_{\mathcal{N}(i)}^{(t)}])$$

where $W^{(t)}$ is the weight matrix of layer t, $\mathcal{N}(i)$ is the neighborhood set of node i, $\sigma^{(t)}$ is the activation function of layer t. Activation function for the first layer is ReLU. It is one of the most commonly used activation functions for modern neural networks. For the last layer, softmax activation function is used to output probabilities for each class.

For this study, most important advantage of using graph-based models is being able to use unlabeled examples. There are almost 20K unlabeled example in the dataset. Graph models are capable of using connections between inputs and features of the neighboring nodes during prediction. This makes sense because people are usually connected with people sharing similar interests. Classical models mostly assume all examples are i.i.d and only take into account the input features.

Table 1. Best results of 5-Fold cross-validation for classical ML models

Model	Weighted F-1
#1 Logistic Regression	0.752± 0.009
#2 Naive Bayes	0.736± 0.007

4. Experiments

For all experiments, 5-fold cross-validation is applied and mean and standard deviation of weighted F-1 scores are reported. Experiments that have produced the best result are reported for all models.

4.1. Classical ML Models

Best results are reported on Table 1. The experiment "#1 Logistic Regression" produced best F-1 score for all logistic regression experiments. It uses both the original features and the mean of the features of neighboring users (follower and following) for a given user. Then it applies feature selection using the feature importances of Extra Trees Classifier. Finally, it trains a one vs. rest logistic regression model on the selected features.

On the other hand, best naive bayes model is given as "#2 Naive Bayes". It also utilizes both the original and mean aggregation of features of neighbors. Then, naive bayes model is trained on all of the features.

4.2. Graph Models

Best results are reported on Table 2. "#3 GraphSAGE" experiment have produced the best result amongst all experiments throughout the study. Before constructing the graph, feature selection is applied using Extra Trees Classifier. After feature selection, an undirected graph is constructed, utilizing both follow and following information. This is logical because of the fact that, a person is usually followed by people who shares common interests just as that person follows similar people to him/her. Also, using the undirected graph allows us to leverage more information. For most cases, this helps increasing the scores specially when we have a small dataset. Two layers of GraphSAGE is used with hidden dimension of 200.

Second most successful experiment is "#4 GCN". It also uses the undirected version of the original graph and the Extra Trees Classifier as feature selection method. The model has 2 layers of graph convolution with hidden dimension 400.

All graph models trained for maximum epochs of 500 with early stopping with patience 50 epochs. Adam, adaptive optimization algorithm, is used with learning rate of 10^{-3}.

Table 2. Best results of 5-Fold cross-validation for graph models

MODEL	WEIGHTED F-1
#3 GRAPHSAGE	**0.762± 0.008**
#4 GCN	0.758± 0.006

5. Discussion & Future Work

In this study, it has been shown that different approaches can be applied to job title prediction of users on a social platform called Github. Each step of the pipeline is given in detail. Results of various experiments are reported.

There are many different design choices for each step of the pipeline. Different preprocessing, feature extraction, feature selection techniques and models could be used. For the current results, it seems that there is not much difference between classical machine learning models and graph deep learning models. This could be caused by many different things. First obvious one is having a small dataset. Another reason may be noisy results of the labeling method. Using the full graph of all Github database and employing more sophisticated labeling techniques could show the power of graph neural networks.

Also, all information of the users in the dataset could not be leveraged due to time constraints. Using full information of all repositories of users might increase the performances. Also, using data of Github stars, issues and pull requests in feature extraction step can help.

6. Code

Code developed during the study can be found here: https://github.com/tugrulhkarabulut/github-job-title-prediction. You can follow the instructions to reproduce the results.

References

API, G. R. Github rest api. https://docs.github.com/en/rest. Accessed: 2022-12-31.

Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr 2006. ISSN 1573-0565. doi: 10.1007/s10994-006-6226-1. URL https://doi.org/10.1007/s10994-006-6226-1.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017. URL http://arxiv.org/abs/1706.02216.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

MultinomialNB. Multinomialnb. https://scikit-learn.org/stable/modules/generated/sklearnṅaive_bayesṀultinomialNBḣtml. Accessed: 2022-12-31.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. *CoRR*, abs/1909.13021, 2019. URL http://arxiv.org/abs/1909.13021.