Github Repository Classifier

Rami Aly[1], **Andre Schurat**[2]

[1] University of Hamburg

[2] Technical University of Dortmund

January 13, 2017

1 Abstract

Contents

2 Selecting features

3 Gathering selected features from Github

4 Removing irrelevant information from selected features

5 Building the Prediction Model

5.1 Choosing a prediction Model

Of course there are many different approaches to the problem. A static algorithm to classify repositories is rather impractical because the parameters of our classify function would be strongly influenced by our interpretation of weights of the features for each class. However we quickly noticed that the complexity is very high, so that a normal algorithm must limit the aspects which can be considered. Above all the problem is non-linear and through the static analysis we would loose the possibility to freely improve or change the classifier. As the software and use-case market of Github rises the possible need of further classes could arise.
Hence to ensure a classifier who is as dynamic and as extensible as possible we choose to use some form of machine learning. The problem which needs to be solved by the Prediction Model is a classification problem: We have a fixed number of values for selected features as input and as an output the class to which the values fit the most. As a result of this fact it was pretty clear to us that a supervised learning method would be optimal.
In the next step we thought about the pro- and contra arguments of non-parametic and parametic learning. For example Gaussian-Process-Models could be used in principle, as one does not need to specify a fixed number of parameters and therefore be non-parametic. The main problem with Gaussian-Process-Models is that they scale rather poorly with a complexity of $O(n^3)$ [1]. Moreover if we keep the huge dimension of repository datasets in mind and as such the possible complexity of the classifier function, our choice will lead us to a parametric neural Network.

5.2 Our Neural Network Model

First of all for our selected features it is not needed to consider temporal behavior. We do need need to save an internal state (If we had used a sequence of commits this could have been otherwise). It should be fully sufficient to use a Feed-forward neural network. As for a Neural Network with supervised learning we choose that a Multilayer perceptron(MLP) with one hidden Layer would be sufficient for our classification problem. We already mentioned that our classification problem is not linear. Hence at least one hidden Layer is required to solve the problem. Furthermore we know that this MLP can approximate any bounded continuous function with arbitrary precision [2], particularly our classification problem. The usage of a linear activation function would result in a MLP with a set of possible functions to be equivalent those of a normal input-output perceptron. Therefore we need to use a nonlinear activation function. We therefore decided to use a Sigmoid function. For the training we used as expected for a MLP Backpropagation. To reduce the chance to be stuck in a local minimum we used intertia so that the previous change will influence the weight adjustment in the current round. The neuron count for the output-layer is set to the number of different classes into which we want to classify the input. So for our Neural Network we used 7 output neurons. Our input neurons count equals to sum of the length of every dictionary plus all relevant ratios of our selected features.

5.3 Format the preprocessed Features to fit into the Neural Network

6 Training Set

7 Optimizing our Neural Network

8 Validation of created Classifier

9 Extensions

References

[1] , David P. Williams Gaussian Processes (2006) http://people.ee.duke.edu/~lcarin/David1.27.06.pdf

[2] Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems http://deeplearning.cs.cmu.edu/pdfs/Cybenko.pdf

10 Source Code and used external Libraries

Source Code Github: https://github.com/Crigges/InformatiCup2017