XXI ENMC
ENCONTRO NACIONAL DE MODELAGEM COMPUTACIONAL
IX ECTM
ENCONTRO DE CIÊNCIA E TECNOLOGIA DE MATERIAIS

08 a 11 de Outubro de 2018
Instituto Federal Fluminense
Búzios - RJ

CLASSIFICAÇÃO DE LINGUAGENS DE PROGRAMAÇÃO UTILIZANDO TÉCNICAS DE *MACHINE LEARNING*

Gabriela Ramalho[1] - gabriela.ramalho@decom.cefetmg.br
Marco Sousa[1] - marco.sousa@decom.cefetmg.br
Bruno Andre Santos[1] - bruno@decom.cefetmg.br
Rogério Martins Gomes[1] - rogerio@cefetmg.br
Renan Mateus B. do Nascimento[1] - renanmateusbn@gmail.com
[1]Centro Federal de Educação Tecnológica de Minas Gerais , DECOM - Belo Horizonte, MG, Brazil

Resumo.

O compartilhamento de programas em diferentes linguagens de programação tem se tornado cada vez mais comum na internet. Reconhecer a linguagem de programação do trecho de código compartilhado é de grande importância para os mecanismos de busca e também na aplicação da formatação adequada. No entanto, a grande maioria dos sistemas realiza esta categorização de forma manual, com base na extensão do arquivo. Sendo assim, este artigo propõe um método automático de classificação de linguagens de programação. O método é composto por duas etapas. Na primeira etapa, uma base de dados consistindo do número de ocorrências das palavras encontradas nos trechos de código é gerada automaticamente. Na segunda etapa, os classificadores Naive Bayes, Redes Bayesianas, Árvore de decisão (J48) e Redes Neurais Artificiais de Múltiplas Camadas são utilizados para realizar a classificação dos trechos de códigos. Os resultados encontrados mostraram que a acurácia alcançada por todos classificadores foi superior a 95,4%, mas que o desempenho é dependente da maneira como a base de dados de palavras é construída. Além disso, este trabalho mostrou que as árvores de decisão J48 apresentaram o melhor resultado, alcançando uma acurácia de 98,9% no processo de classificação das linguagens de programação.

Keywords: *Classificadores, Linguagens de Programação, Aprendizado de Máquina*

1. INTRODUÇÃO

Diariamente, uma enorme variedade de programas e pedaços de códigos são desenvolvidos e adicionados a inúmeros websites e bases de dados na internet. Como exemplos, pode-se destacar os repositórios de código (*GitHub* e *Bitbucket*), os grupos de discussão (*StackOverflow*), os sistemas de rastreamento de defeitos e melhorias (*Jira*), os sistemas de compartilhamento de código (*Pastebin* e *GitHub Gists*), dentre outros, que têm sido desenvolvimentos

com o objetivo de permitir o desenvolvimento, o compartilhamento, a manipulação e o gerenciamento do processo de desenvolvimento de *software*.

Para todas estas fontes de programas, a tarefa de reconhecimento da linguagem de programação do código que está sendo publicado pelo usuário tem grande relevância. O reconhecimento da linguagem, por exemplo, permite a identificação da formatação a ser aplicada de acordo com a sintaxe especificada, além de possibilitar a categorização de um determinado trecho de código, auxiliando, dessa forma, na implementação de mecanismos de busca e no levantamento estatístico que pode ser utilizado na identificação de tendências. Em muitos casos, a identificação é realizada tendo como base a extensão do arquivo ou também pelo usuário que, muitas das vezes, registra explicitamente nos metadados do projetos a linguagem do programa usado. A desvantagem desses métodos é que eles podem não ser adequados para classificação de programas em grandes bases de dados nas quais os usuários não realizam os registros de maneira adequada e, principalmente, nos casos em que o trecho de código compõem um arquivo com outros conteúdos, como por exemplo em páginas *html* ou em documentos no formato PDF (*Portable Document Format*).

Este trabalho tem como objetivo aplicar e analisar métodos de *machine learning* na classificação automática de linguagens de programação. Os métodos utilizados são: *Naive Bayes*, Redes Bayesianas, Árvore de decisão (J48) e Redes Neurais Artificiais de Múltiplas Camadas (Haykin, 2009). A base de dados de linguagens é criada de forma automática, independente da sintaxe da linguagem. Dessa forma, o esforço para se acrescentar suporte a novas linguagens será minimizado, ao contrário do que é visto em outros trabalhos publicados na literatura, nos quais a identificação da linguagem é orientada a sintaxe (vaDam, 2016; Ugurel, 2002; Khasnabish, 2014; klein, 2011; Zevin and Holzem, 2017).

Este artigo está organizado da seguinte forma: a Seção 2 apresenta a metodologia utilizada na construção da base de dados a ser utilizada, bem como os classificadores usados na identificação das linguagens de programação. A Seção 3 apresenta e analisa os resultados obtidos e, finalmente, a Seção 4. conclui o artigo e apresenta perspectivas de trabalhos futuros.

2. METODOLOGIA

Um sistema completo de reconhecimento de padrões é composto por algumas etapas. A primeira etapa é constituída pela criação da base de dados. A seguir, os dados obtidos da etapa anterior são enviados aos classificadores que, utilizando algum algoritmo ou técnica computacional, define a classe referente dos dados de entrada. A Fig. 1 mostra o organograma completo com todas as fases dos sistema proposto.

2.1 Criação da Base de Dados

A primeira etapa, de criação da base de dados, foi dividida em 4 passos. No primeiro passo, escolheu-se dez linguagens de programação mais populares segundo o índice TIOBE (TIOBE (2018)): Java, C, C++, Pyhton, C#, Visual Basic, Javascript, PHP, Ruby e Pascal. Para cada linguagem, escolheu-se três projetos de código aberto dentre os cinco mais populares, em número de *stars*, do GitHub. Todos os arquivos com extensão conhecida contidos nesses projetos foram separados de acordo com a Tabela 1. Seguem os projetos selecionados:

- Java:
  i) Design patterns implemented in Java (https://github.com/iluwatar/java-design-patterns)

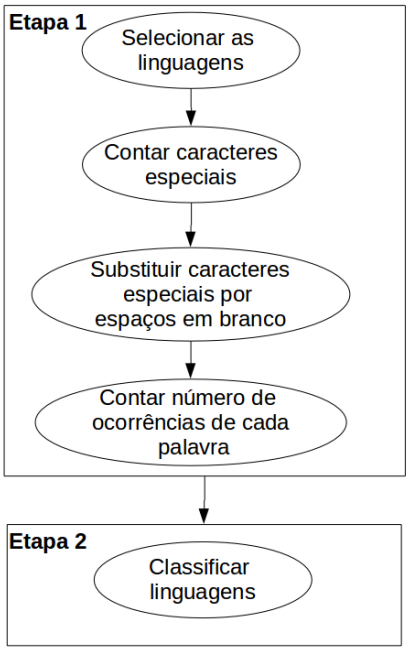

Figure 1- Etapas e fases do processo classificação de linguagens de programação. Na primeira etapa (retângulo superior), a base de dados com as diversas linguagens de programação é construída. Na segunda etapa (retângulo inferior), os métodos computacionais são utilizados para classificar as linguaguens de programação.

ii) RxJava: Reactive Extensions for the JVM (https://github.com/ReactiveX/RxJava)
iii) Elasticsearch (https://github.com/elastic/elasticsearch)

- C:
  i) Netdata (https://github.com/firehol/netdata)
  ii) Redis (https://github.com/antirez/redis)
  iii) Git (https://github.com/git/git)

- C++:
  i) TensorFlow (https://github.com/tensorflow/tensorflow)
  ii) Electron framework (https://github.com/electron/electron)
  iii) Swift (https://github.com/apple/swift)

- *Python:*
  i) Youtube-dl (https://github.com/rg3/youtube-dl)
  ii) TensorFlow Models (https://github.com/tensorflow/models)
  iii) Flask (https://github.com/pallets/flask)

- C#:
  i) Shadowsocks for Windows (https://github.com/shadowsocks/shadowsocks-windows)
  ii) CodeHub (https://github.com/CodeHubApp/CodeHub)
  iii) .NET Core Libraries https://github.com/dotnet/corefx)

- *Visual Basic:*

i) CompactGUI (https://github.com/ImminentFate/CompactGUI)
ii) VBA-Web (https://github.com/VBA-tools/VBA-Web)
iii) StaxRip (https://github.com/stax76/staxrip)

- *JavaScript:*
  i) freeCodeCamp (https://github.com/freeCodeCamp/freeCodeCamp)
  ii) React (https://github.com/facebook/react)
  iii) Vue.js (https://github.com/vuejs/vue)

- *PHP*:
  i) Symfony PHP framework (https://github.com/symfony/symfony)
  ii) CodeIgniter (https://github.com/bcit-ci/CodeIgniter)
  iii) Composer (https://github.com/composer/composer)

- *Ruby:*
  i) Rails (https://github.com/rails/rails)
  ii) Jekyll (https://github.com/jekyll/jekyll)
  iii) Homebrew (https://github.com/Homebrew/legacy-homebrew)

- *Pascal:*
  i) RDP Wrapper Library (https://github.com/stascorp/rdpwrap)
  ii) Cheat Engine (https://github.com/cheat-engine/cheat-engine)
  iii) Inno Setup (https://github.com/jrsoftware/issrc)

Table 1- Linguagens de programação selecionadas

Linguagem	**Extensão**	**Número de arquivos**
Java	.java	122
C	.c	1113
C++	.cpp	755
Python	.py	5398
C#	.cs	14655
Visual Basic	.vb	158
JavaScript	.js	1624
PHP	.php	4811
Ruby	.rb	2775
Pascal	.ps	556
Total		31967

Para exemplificar os próximos passos da etapa 1, foi tomado como modelo o trecho de código a seguir:

```
int main(){
    printf("Hello World");
}
```

Após a seleção das linguagens, no segundo passo da Etapa 1, foi realizada a contagem do número de caracteres especiais (não alfanuméricos) em cada um dos arquivos selecionados. Para o modelo tomado de exemplo:

(	)	”	{	}	;
2	2	2	1	1	1

A seguir, no terceiro passo, todos os caracteres especiais encontrados no código são substituídos por espaços em branco:

```
int main
    printf  Hello World
```

Ao final, a frequência de todas as palavras restantes no arquivo é calculada:

int	main	printf	Hello	World
1	1	1	1	1

As palavras mais comuns de cada linguagem foram selecionadas para servirem de atributos à base de dados, sendo o limiar definido no momento da geração dessa base de dados. Um limiar de 0.1 significa que apenas as palavras (ou símbolos não alfanuméricos) com frequência maior que 10% para cada linguagem serão selecionadas. Sendo assim, pode-se perceber, pela Tabela 2 e Fig. 2, que o número de atributos cresce a medida que um limiar menor é escolhido.

Table 2- Variação do limiar e características da base de dados

Limiar	**Número de Atributos**	**Tamanho do arquivo**
0.1	2	1,3M
0.05	11	6,2M
0.01	40	18M
0.005	77	28M
0.003	129	44M
0.001	348	94M

2.2 Classificadores

Após a criação da base de dados com a frequência de palavras encontradas, passou-se para a última etapa, na qual foi utilizado o módulo classificador. Para simular este módulo, foi utilizada a ferramenta *Weka*, abreviação para *Waikato Environment for Knowledge Analysis*. O *Weka* é um *software* criado pelo grupo de aprendizado de máquina da Universidade de Waikato (Witten et al., 2016), que dispõe de diversos algoritmos de aprendizado de máquina. Os classificadores usados neste artigo foram: *Naive Bayes*, Redes Bayesianas, Árvore de decisão (j48) e as Redes Neurais Artificiais de Múltiplas camadas.

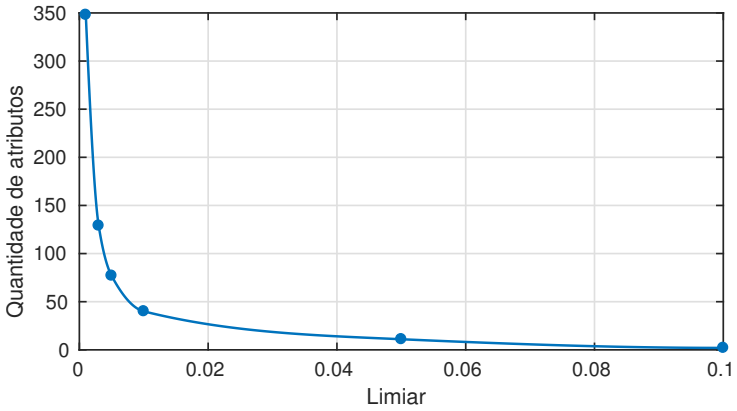

Figure 2- Valor do limiar de seleção x número de atributos.

Naive Bayes: O classificador Naive Bayes provê uma simples abordagem para representar, usar e desenvolver um conhecimento probabilístico. Se comparado às redes bayesianas, é mais simples ao assumir que os atributos são condicionalmente independentes. Essa suposição tem como objetivo permitir o desenvolvimento de algoritmos eficientes para classificação como também para treinamento (John and Langley, 1995). Temos que, pelo Teorema de Bayes:

$$p(C = c|X = x) = \frac{p(C = c)p(X = x|C = c)}{p(X = x)}, \tag{1}$$

em que:

$$X = x \iff X_1 = x_1 \wedge X_2 = x_2 \wedge ...X_k = x_k \tag{2}$$

O parâmetro C é uma variável aleatória denotando a classe de uma instância; X é um vetor de variáveis aleatórias denotando os valores dos atributos; c é a representação particular de uma classe (de uma linguagem de programação); e, por fim, x é o valor de um atributo em particular (de uma palavra ou de um caractere especial).

Segundo John and Langley (1995), o resultado da suposição dos atributos como condicionalmente independentes implica que:

$$P(X = x|C = c) = p(\bigwedge_i X_i = x_i|C = c) = \prod_i p(X_i = x_i|C = c) \tag{3}$$

A Equação 3 é simples de computar para casos de testes e simples de estimar para o conjunto de treinamento. Neste trabalho, foi adotada a configuração padrão do método e utilizou-se 66% das instâncias do base de dados para o treinamento e as demais instâncias como casos de teste.

Redes Bayesianas: As redes bayesianas são grafos acíclicos direcionados que representam a distribuição de probabilidade conjunta sobre um conjunto de variáveis aleatórias. Essa representação é um aprimoramento do *Naive Bayes*, sendo que nessa abordagem a correlação entre os atributos não é ignorada. Cada vértice do grafo representa uma variável aleatória e as arestas definem a correlação entre as variáveis (Friedman et al., 1997). Para este trabalho, utilizou-se as redes bayesianas para classificação, utilizando-se de 66% das instâncias da base de dados para treinamento e as demais instâncias para avaliação.

Árvore de Decisão: Uma árvore de decisão recebe como entrada um objeto ou situação descrita por um conjunto de propriedades e retorna uma decisão binária. Cada nó da árvore corresponde à um teste de um valor de uma propriedade. Cada folha na árvore corresponde ao valor a ser retornado caso a folha seja alcançada (Russel and Norvig, 1995). Para uma árvore de decisão, tem-se como parâmetros: o número mínimo de instâncias por folha, sendo o valor escolhido para esse trabalho igual a 2, uma vez que somente é possível realizar a comparação de no mínimo duas linguagens distintas; e o parâmetro de confiança de pós-poda, que foi definido como 0,25. Este valor foi escolhido empiricamente e tem relação com a estimativa de erro de cada nó. Assim, para um fator de confiança menor nos dados de treinamento, a estimativa de erro para cada nó aumenta, aumentando a probabilidade de do mesmo ser removido.

Redes Neurais Artificiais de Múltiplas Camadas: Inspirada no cérebro humano, uma rede neural é uma composição de unidades (ou nós) conectadas entre si com um peso em cada uma de suas conexões. As saídas de cada uma das unidades é relativa às suas entradas, numa proporção chamada de nível de ativação. Dessa forma, cada unidade realiza uma computação local baseada nas entradas advindas de unidades vizinhas e sem dependência de um controle global, conforme representação mostrada na Fig. 3 (Russel and Norvig, 1995).

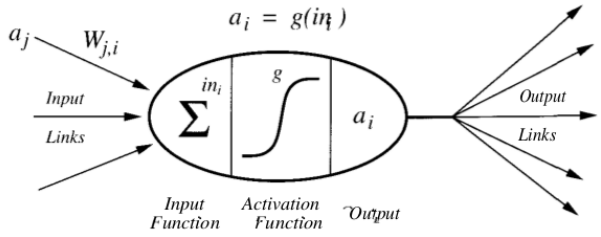

Figure 3- Representação gráfica de uma unidade de uma rede neural (Russel and Norvig, 1995)

Cada unidade de uma rede neural realiza, na sua primeira etapa, a soma ponderada dos valores de suas entradas (uma componente linear). Ou seja, para uma unidade i, tem-se que seu valor final de entrada in_i é dado pela Eq. 4:

$$in_i = \sum_j W_{j,i} a_j \tag{4}$$

em que $W_{i,j}$ é o peso de uma conexão j para esta mesma unidade i; e a_j é o valor de entrada advindo de uma outra unidade por esta conexão j.

Segundo Russel and Norvig (1995), a segunda etapa é a aplicação do valor final de entrada numa função de ativação g (uma componente não linear) responsável por transformar a soma ponderada em um valor final a_i que, por fim, será utilizado como entrada para outras unidades. Para este trabalho, utilizou-se a função ativação sigmoidal, devido ao fato da mesma fornecer a probabilidade de cada entrada pertencer a uma determinada classe.

Para se construir uma rede neural, é necessário decidir alguns parâmetros como quantas unidades (nós) serão utilizadas e como as unidades serão conectadas entre si. Para este trabalho, decidiu-se utilizar uma rede de várias camadas, sendo o número de camadas escondidas (desconsiderando a primeira e última) foi definido por $\frac{atributos+classes}{2}$, em que o número de classes é o número de linguagens de programação e o número de atributos varia de acordo com a Tabela 2; utilizou-se da técnica de *backpropagation* para cálculo dos pesos (treinamento), o

que significa que erros em unidades de camadas posteriores afetam unidades de camadas anteriores. O número de camadas, bem como o algoritmo de treinamento usado foram os que apresentaram os melhores resultados.

Outros parâmetros, característicos das redes neurais, também foram ajustados de forma empírica, e os valores que apresentaram os melhores resultados foram os seguintes: taxa de aprendizado igual a 0,3; taxa de momento igual a 0,2; e número de épocas igual a 500. Para realização do treinamento da rede neural, utilizou-se 66% das instâncias geradas. As demais, por sua vez, foram utilizadas para avaliação.

3. RESULTADOS

Testes foram realizados com a base de dados criada para cada um dos valores de limiar da Tabela 2 e para cada um dos algoritmos utilizados como classificadores. A Tabela 3 mostra a acurácia alcançada nos experimentos.

Table 3- Acurácia da classificação - Limiar *versus* classificadores

Limiar	*Naive Baye*s	Redes Bayesianas	J48	Redes Neurais Artificiais
0,1	65,1%	65,6%	68,9%	68,0%
0,05	78,2%	88,2%	92,1%	90,1%
0,01	87,4%	95,7%	98,1%	97,4%
0,005	88,3%	95,8%	98,1%	97,6%
0,003	95,6%	95,4%	98,3%	98,1%
0,001	97,6%	94,8%	98,9%	60,8%

Na Fig. 4 observa-se que ao diminuir o limiar, a tendência é aumentar a acurácia. Já a acurácia das Redes Neurais Artificiais decaem a partir do limiar 0,003. Isso ocorre pois, para um limiar alto, o número de atributos é muito restritivo, sendo insuficiente para as redes neurais conseguirem classificar de maneira correta. Por outro lado, para um limiar muito baixo, uma grande quantidade de atributos é levada consideração, incluindo aqueles que não são significativos, ou seja, que não caracterizam a linguagem de programação, dificultando o processo de classificação. Por este motivo, a acurácia para o limiar 0,1 é mais baixa.

4. CONCLUSÃO

Neste trabalho foram utilizados quatro de *machine learning* para classificação de linguagens de programação. Dois pontos podem ser destacados como contribuição deste artigo. O primeiro ponto diz respeito à metodologia para construção da base de dados de forma automatizada e independente da linguagem de programação. Na metodologia, é importante ressaltar a influência do parâmetro *limiar* para o cálculo da acurácia de classificação dos métodos. Nas Redes Neurais Artificiais, por exemplo, ao aumentar o limiar de 0,003 para 0,001 (ou seja, ao aumentar o número de símbolos da base de dados), a acurácia caiu de 98,1% para 60,8%. O segundo ponto diz respeito aos resultados obtidos pelos classificadores. Observou-se que a árvore de decisão (J48) apresentou o melhor resultado, alcançando uma acurácia de até 98,9% na classificação das linguagens de programação. Ao utilizar o limiar de 0,003, a acurácia mais baixa foi de 95,4%. Dessa forma, os resultados obtidos sugerem que a utilização de métodos de *machine learning* podem auxiliar positivamente na classificação de linguagens de programação.

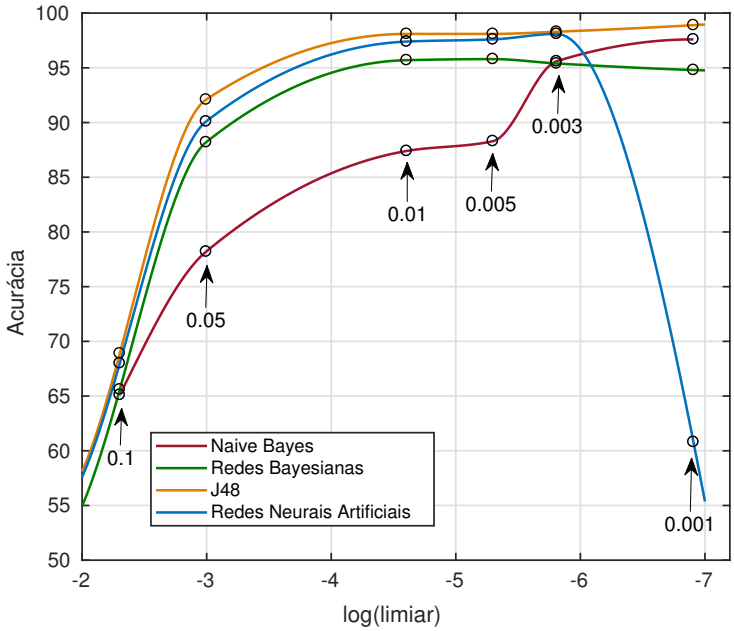

Figure 4- Acurácia (eixo y) para diferentes valores de limiar (eixo x). Para uma melhor visualização, está sendo exibido o *log* do limiar (ver eixo x). As setas indicam os valores dos limiares.

AGRADECIMENTOS

Os autores gostariam de agradecer ao CEFET-MG e à FAPEMIG pelo suporte financeiro, sem o qual esse trabalho não teria sido viável.

REFERÊNCIAS

Duda, Richard O; Hart, Peter E; Stork, David G (2012), *Pattern classification*, John Wiley & Sons.

Haykin, S. O. (2009), *Neural networks and learning machines*, 3ª ed., Pearson, Upper Saddle River, NJ, USA.

Khasnabish, J. N., Sodhi, M., Deshmukh, J., and Srinivasaraghavan, G. (2014), Detecting programming language from source code using bayesian learning techniques. *In International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 513–522. Springer.

Klein, D., Murray, K., Weber, S. (2011) Algorithmic programming language identification, arXiv preprint arXiv:1106.4064.

Ugurel, S., Krovetz, R., and Giles, C. L. (2002). What's the code?: automatic classification of source code archives. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 632–638. ACM.

TIOBE Index (2018), Very Long Term History, https://www.tiobe.com/tiobe-index/, acessado em 04-06-2018.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016), Data Mining: Practical machinelearning tools and techniques. Morgan Kaufmann.

van Dam, J. K. and Zaytsev, V. (2016). Software language identification with natural language classifiers. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 624–628. IEEE.

Zevin, S. and Holzem, C. (2017), Machine learning based source code classification using syntax oriented features.arXiv preprint arXiv:1703.07638.

Russell, Stuart J. and Norvig, Peter (1995). Artificial Intelligence: A Modern Approach. Prentice-Hall.

John, G. H. and Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. In *11th Conference on Uncertainty in Artificial Intelligence*, pages 338-345.

Friedman, N., Geiger, D. and Goldszmidt, M. Machine Learning (1997) 29: 131. https://doi.org/10.1023/A:1007465528199

CLASSIFICATION OF PROGRAMMING LANGUAGES USING MACHINE LEARNING TECHNIQUES

Abstract. *Sharing programs in different programming languages has become increasingly common on the internet. Recognizing the programming language of the shared code snippet is of great importance for the search engines and also in the application of the appropriate text formatting. However, most systems perform this classification manually, based on the file extension. Thus, this article proposes an automatic method for the classifying programming languages. The method consists of two steps. In the first step, a database consisting of the number of words found in the selected language is automatically generated. In the The second step, the computational methods Naive Bayes, Bayesian Networks, Decision Tree (J48) and Artificial Neural Networks of Multiple Layers are used to perform the classification of code snippets. The results showed that the accuracy of all classifiers was greater than 95.4 %, but that performance is dependent on the way the database is build. In addition, the results showed that the J48 decision trees presented the best result, reaching an accuracy of 98.9 % in the classification process of programming languages.*

Keywords: *Classifiers, Programming Languages, Dataset generator*