Authorship Identification of Source Codes Written by Multiple Authors Using Stacking Ensemble Method

Parvez Mahbub

Naz Zarreen Oishie

Computer Science and Engineering Discipline

Khulna University

Khulna – 9208, Bangladesh

Authorship Identification of Source Codes Written by Multiple Authors Using Stacking Ensemble Method

Parvez Mahbub

Student ID: 150204

Naz Zarreen Oishie

Student ID: 150201

Computer Science and Engineering Discipline

Khulna University

Khulna – 9208, Bangladesh

Khulna University
Computer Science and Engineering Discipline

The undersigned hereby certify that Naz Zarreen Oishie and Parvez Mahbub have read and recommended to the Computer Science and Engineering Discipline for acceptance of a thesis entitled "Authorship Identification of Source Codes Written by Multiple Authors Using Stacking Ensemble Method" in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering (CSE).

Date: January 23, 2019

Dr. S.M. Rafizul Haque — Thesis Supervisor
Professor
Computer Science and Engineering Discipline
Khulna University

Dr. Kamrul Hasan Talukder — Second Examiner
Professor
Computer Science and Engineering Discipline
Khulna University

Dr. Md. Anisur Rahman — Head of the Discipline
Professor
Computer Science and Engineering Discipline
Khulna University

Acknowledgement

First of all, we would like to thank Almighty Allah for giving us mental and physical strength to complete our thesis properly. We would like to express our sincere gratitude to our supervisor Dr. S.M. Rafizul Haque, Professor, Computer Science and Engineering Discipline, Khulna University, for his co-operation, suggestion, guidance, and continuous encouragement through the course of the study.

We are grateful to our second examiner Dr. Kamrul Hasan Talukder, Professor, Computer Science and Engineering Discipline, Khulna University, for reviewing our thesis and giving us thoughtful suggestions to improve it.

Along with our thesis supervisor and second examiner we would like to especially thank Dr. Manishankar Mondal, Assistant Professor, Computer Science and Engineering Discipline, Khulna University, for guiding us and providing continuous feedback with his excellence in the discipline of software engineering.

We would like to acknowledge the support of our honorable teachers of the discipline throughout the path with their encouragement and suggestions. We also thank our parents for their encouragement, support and always being the source of our moral strength.

Abstract

Source code authorship identification is the issue of identifying the author of a source code based on the experience of previous source codes. It has vast importance in plagiarism detection, digital forensics, and several other law enforcement issues. However, when the number of writers of a source code is more than one, typical author identification systems no longer work. In this thesis, we have implemented a source code author identification system using stacking ensemble classifier that can predict the authorship of source codes even when the number of authors is more than one. Our stacking ensemble system is built upon several deep neural networks, random forests and support vector machine classifiers. We have shown that when identifying the author in such case, a single classification technique is no longer sufficient and using a deep neural network based stacking ensemble method can increase the accuracy significantly. We have compared our work with several other works that only deals with source codes that are written by exactly one author. The experimental result shows that even after using source codes written by multiple authors, we have achieved accuracy pretty close to the related works.

Keywords: Source Code Authorship Identification, Multiple Author, Deep Neural Network, Random Forest, Support Vector Machine, Stacking Ensemble

Contents

1 Introduction **1**

1.1 Problem Definition 2

1.2 Motivation 3

1.3 Objective 5

1.4 Chapter Outline 5

2 Background **7**

2.1 Machine Learning 7

2.2 Feature Extraction 9

2.2.1 Code Metric 9

2.3 Normalization 10

2.4 Deep Neural Network 10

2.4.1 Activation Function 11

2.4.2 Loss Function 12

2.4.3 Optimizer 13

2.4.4 Overfitting 14

2.5 Random Forest 16

2.6 Support Vector Machine 17

2.7 Ensembling Methods 19

2.7.1 Stacking Ensemble Method 19

3 Related Works **21**

3.1 Text Based Approaches 21

3.2 Metric Based Approaces 22

4 Author Identification of Source Codes Written by Multiple Authors **24**

4.1 Dataset 26

4.2 Metric Extraction 27

4.3 Base Classifiers 29

4.3.1 Deep Neural Network 30

4.3.2 Random Forest 31

4.3.3 Support Vector Machine 31

4.4 Meta Classifiers 31

4.5 Training 33

5 Experimental Results **35**

5.1 Experimental Setup 35

5.1.1 Toolset 35

5.1.2 Parameters 36

5.2 Metrics 36

5.2.1 Accuracy 36

5.2.2 F1-Score 36

5.3 Results of The Base Classifiers 37

5.4 Results of The Meta Classifier 37

6 Conclusion **40**

6.1 Future Direction 41

List of Tables

4.1 Number of authors and contributors for each class 27
4.2 Set of code metrics and descriptions 28

5.1 Accuracy of the base classifiers 38
5.2 The confusion matrix of for the testing dataset 38
5.3 Comparison among the methods for source code author identification 39

List of Figures

2.1 Training phase of a machine learning algorithm 8

2.2 Testing phase of a machine learning algorithm 9

2.3 High level overview of a deep neural network 11

2.4 Stochastic gradient descent algorithm 13

2.5 Comparison of Adam with other optimizers 14

2.6 Adam algorithm 15

2.7 Pictorial illustration of how regularization prevents overfitting 16

2.8 Overview of random forests 17

2.9 Overview of support vector machine 18

2.10 Overview of stacking 20

2.11 Stacking algorithm 20

4.1 Block diagram of the architecture of the stacking ensemble method . 25

4.2 Block diagram of proposed author identification approach 26

4.3 Block diagram of the neural network used as base classifier 30

4.4 Block diagram of the neural network used as meta classifier 32

4.5 Steps for training the stacking ensemble system 33

Chapter 1

Introduction

Source code author identification is a major research topic in the field of software forensics. It has many uses such as plagiarism detection, law enforcement, copyright infringement etc.[1, 2]. Zobel[3] has mentioned that selling solution of assignments to multiple students is a common source of plagiarism. This can be stopped if all of the assignments from the same source are identified by the same author. Frantzeskou[4] mentioned that source code author identification is useful against cyber attacks in the form of viruses, trojan horses, logic bombs, fraud, credit card cloning, and authorship disputes or proof of authorship in court. There are certain patterns that developers sub-consciously reflect in their codes based on their particular coding style while still following the guidelines, standards, rules, and grammars of a language or framework[2]. These pieces of information can be used to identify the author of the source code.

In recent years, open source software development has entered a new era. A lot of big companies like Google, Microsoft, and many others are maintaining their projects open source. Alongside, small and mid-level projects are being written by a group of authors. In these cases, trivial author identification schemes no longer work. When someone contributes to an open source project, the writing style of the original author of the source code is no longer unique and it makes the author identification task harder. Even worst case is when a project is equally contributed by a number of authors. The writing style is then the aggregation of all the authors. We aim to solve this problem and make an author identification system that can identify the author of a source code even when it is contributed by more than one author.

1.1 Problem Definition

Authorship identification is the task of having some samples of code for several programmers and determining the likelihood of a new piece of code having been written by each programmer[5]. As the name suggests, authorship identification of source codes written by multiple authors is identifying the author-label when the number of authors of source code is more than one. This can be of two kinds. Firstly, the source code can be written by mostly one author and have small contributions from several other authors. This is what happens in typical open source projects. Alternatively, source codes can be directly written by a group of authors and have a roughly equal contribution from each of them. This is what happens when a team is assigned to a project and the members work in a collaborative fashion.

Typical author identification tasks often assume each of the source codes is written by exactly one author. However, nowadays a vast amount of software and projects are being maintained open source. This enables a single source code to be contributed by a large number of authors. In the case of contributions, the main author has a unique writing style and when some other person contributes to the code, the writing style deviates from the original author. As a result, author identification becomes harder. Alongside, with the help of several version control systems and team collaboration software like git or SVN, most of the projects are currently being written by a group of authors. In this thesis, we consider this group of authors as a single class-label. As a result, to identify the class-label(author) of source code we need to identify the aggregated writing styles of the group of authors.

Therefore, author identification of a source code written by multiple authors is much harder than typical author identification systems that aim source codes written by only one author.

1.2 Motivation

Authorship identification of source code has a vast application area including plagiarism detection, authorship dispute, software forensics, malicious code tracking, criminal prosecution, software intellectual property infringement, corporate litigation, and software maintenance[1, 6–9].

Author identification techniques can be used to prevent source code plagiarism[10]. By determining if a suspect code really was written by the claimed author, it can be concluded that if that code is plagiarized or not[8]. Students often borrow code from classmates, friends, web or home tutor[3, 7]. If an author identification system is trained using source codes from all authors of an academic institution, a code borrowed from classmates or friends can easily be detected. Even in the case of copying from the web or home tutor, what is impossible to be in the scope of source code sampling, an authorship identification system can check the likelihood of writing a code for a particular author. This can be another approach to detect plagiarism.

In the case of authorship dispute, authorship identification can be a solution. Given the source code and the candidate owners, the likelihood of each candidate of being the author of the source code can be determined[2]. From this, the most likely candidate can be identified as the author of the reference code.

Author identification has a vast use in software forensics. Software forensic is analyzing software to identify the characteristics of the corresponding author for use in forensic activities. Most of the software forensic activities aim to identify the author of a malicious code left in an infected device after a vulnerable attack[7, 9]. Kothari[2] identified that author identification is useful for detecting the author of malicious codes. Given a database of known malware and their authors, when a new malware appears, the likelihood that it is written by an author who previously authored such code can be determined.

Software intellectual property infringement activities like code cloning can be detected using author identification. If a company suspects a former employee of violating the no-compete clause of the contract, they can use author identification to determine if that particular employee wrote the leaked code[8].

Software companies can also use authorship identification system to keep track of programs and modules for better maintenance[9]. A major problem is software industries are maintaining legacy code. Legacy code is known as source code that is no longer supported. So software companies convert it into modern software language and platform. A company with a number of developers, the developer whose coding style matches the most with the original author of the legacy code may be most productive to maintain the code.[8]

It is true that there exist several challenges for source code author identification than that of natural language and speech recognition. However, Frantzeskou[4] opinionated that still being much more restrictive and formal than spoken or written language, source codes inhibit a large degree of flexibility. According to Shevertalov[11], using differences in the way programmers express their idea, their programming style can be captured. This programming style, in turn, can be used for author identification.

Although, a large number of works already done regarding source code author identification, according to Frantzeskou[1], the future of source code author identification is in collaborative projects to which we aim at.

1.3 Objective

We designed a stacking ensemble method based author identification system for source codes written by multiple authors. An ensemble method is a machine learning method that passes the output from several machine learning classifiers, known as base classifiers, to the input of another machine learning classifiers know as meta-classifier. Ensembling methods often show better accuracy than any of the base classifiers.

The main objectives of the work are –

- Finding a better way to predict the class-label(authors) of a new source code written by more than one author.
- Choosing a sufficient machine learning method that can predict the class-label of such source codes.
- Reducing overfitting to the training dataset.
- Keep the system programming language independent.

1.4 Chapter Outline

This thesis is organized into six chapters. Each of them contains different aspects of our thesis. These chapters are briefed below.

Chapter 1: This chapter provides an *introduction* to our thesis which includes the briefing of our work, the definition of the problem we have worked on, the motivation behind our work and the objective of our thesis.

Chapter 2: This chapter contains the *theories and concepts* regarding this work which includes machine learning, deep neural network, support vector machine, random forests, and ensemble methods.

Chapter 3: This chapter contains the *related works* in recent years. Their limitations are also discussed in this chapter.

Chapter 4: In this chapter, the method for *author identification of source codes written by multiple authors* is discussed. How stacking ensemble method can be used for source code author identification and the building blocks of the method are included here. This chapter also includes the details of the dataset that is used in this thesis.

Chapter 5: This chapter contains the *experimental results* of our work. The tools we have used during the implementation of our work is listed here. The comparison of our work with several other related works is also shown here.

Chapter 6: Finally, in this chapter, the *conclusions* and the future directions are enlisted.

Chapter 2

Background

Like all other machine learning applications, source code author identification requires to extract features from samples, split the data into training set and testing set, feed the training set to a machine learning model and train it, and finally evaluate the performance of the model using testing set. Along with all these common features, source code author identification task requires some problem specific tasks as well. Features of the source codes are basically categorized into two types – n-gram and code metric. Extracting them are two completely different processes. Caruana[12] showed that in higher dimension, random forests, deep neural networks(DNN) and support vector machines(SVM) outperform most other machine learning classifiers. However, when a single machine learning classifier does not perform good enough, an aggregation of several classifiers can be used. This technique is called ensembling.

2.1 Machine Learning

Machine learning is a field of artificial intelligence which deals with statistical techniques that give computers the ability to learn from data without being explicitly programmed[13]. Tom Mitchel gave a famous and more formal description of machine learning – "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"[14].

In general, a machine learning algorithm is used to map some input features to the output using a non-linear transformation. For example, we can consider a rice-variant identifier problem. Say, the input features for this problem are the length of the leaf, the width of the leaf and color of the leaf. The output is to which variant of rice the given leaf belongs to. A machine learning algorithm can generate a function that can map the input features to the corresponding output labels. This function is known as a model or trained model. After learning the mapping, the model can even map a label without previous experience of that particular feature set.

Figure 2.1 shows the flow chart of the training phase of a machine learning algorithm. Given the input and output, a machine learning system will extract the features and feed them to the machine learning algorithm along with corresponding labels. The algorithm will output a trained model.

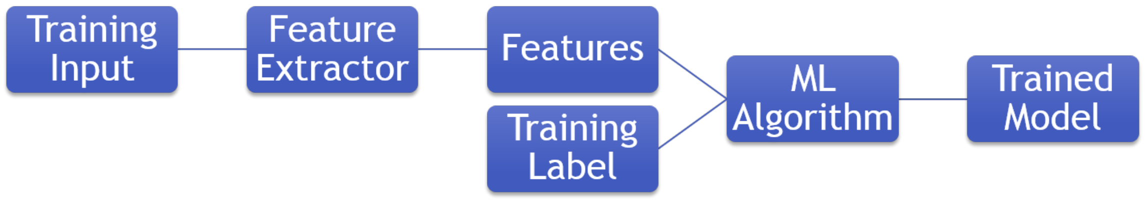


Figure 2.1: Training phase of a machine learning algorithm

Figure 2.2 shows the block diagram of the testing phase of the performance of a machine learning algorithm. In this phase, a separate set of input features are fed to the trained model. The trained model predicts the label for each feature set and the predicted label are matched with the actual label to calculate the accuracy of the model.

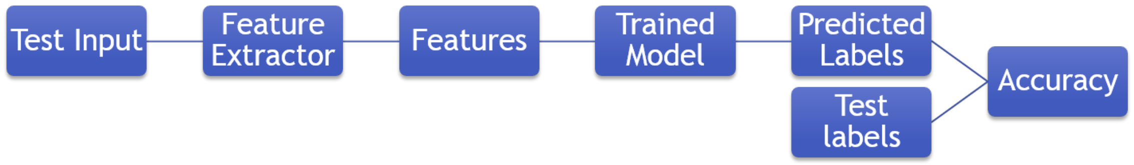


Figure 2.2: Testing phase of a machine learning algorithm

2.2 Feature Extraction

Feature extraction is the first and one of the most important tasks in machine learning. For any input, it is required to extract the proper features that are directly responsible for the corresponding output label. Let's reconsider the rice-variant identification problem. If the input is images of the rice-plants, then firstly it is required to detect the position of the leaves from the images. Later, using the position of the leaves the length, width and color of the leaves can be extracted. Finally, the extracted features will be fed to the machine learning algorithm. A note to remember that for a particular input, a large number of features can be extracted. However, only a very few of them are responsible for the corresponding output label. These characteristics made feature extraction an iterative process which requires several iterations of the whole training phase to find the best feature set.

2.2.1 Code Metric

Source code metric or simply code metric is a measurement that provides a summarized insight of the code. For example, 'inheritance depth' is a source code metric which calculates the depth of inheritance of a particular class. No doubt, it can significantly give an idea of the writing style of the author. Several source code author identification approaches use code metrics as the feature of machine learning algorithm[8, 9, 11, 15].

2.3 Normalization

Normalizing the data attempts to give all attributes an equal weight. Which is to scale the data within a specified range.

In our thesis, we have used Z-score normalization, which is a measure of the number of standard deviation a raw score is far from the mean of the dataset. To perform Z-score normalization, the mean and the standard deviation of the dataset are needed to be calculated. If the mean is μ and the standard deviation is σ of the dataset, then

$$z_i = \frac{x_i - \mu}{\sigma} \tag{2.1}$$

where x is the raw data, z is the normalized value and i denotes the data point. After performing Z-score normalization, the mean of the data becomes 0 and the standard deviation of the data becomes 1.

2.4 Deep Neural Network

The deep neural network is a machine learning algorithm which can be used for classifying data into classes. A deep neural network is composed of an input layer, followed by several hidden layers, followed by an output layer. Figure 2.3 shows a high-level overview of a DNN.

A layer in a neural network is composed of a number of units. A unit is simply a mathematical function that takes input from a number of units in the previous layer, performs a simple mathematical or logical operation and then provides its output to a number of units in the next layer. An input layer is a layer that takes input from an external system and provides input to the first hidden layer. It has the same number of units as the size of the input. A hidden layer is a layer of which each unit is connected to each unit of the previous layer and each unit of the next layer. An output layer simply takes input from the last hidden layer and provides the output of the neural network to an external system. The number of units in an output layer is the same as the number of classes in case of multi-class classification problem and one in case of binary classification or regression problem.

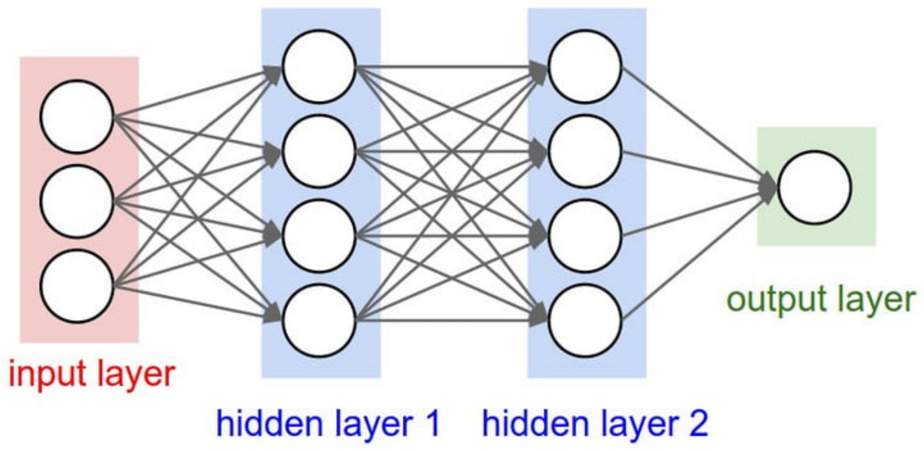


Figure 2.3: High level overview of a deep neural network

2.4.1 Activation Function

Along with a mathematical function, each unit in a DNN optionally performs a logical function known as the activation function. An activation function takes the output of the mathematical function of the unit and outputs a value that can be used as input for the units in the next layer. This output can be either discrete or continuous value.

Rectified Linear Unit(ReLU)

Rectified Linear Unit(ReLU) activation function is a widely used and popular activation function in neural networks. It's primary plus points is its biological similarity, training speed of neural networks, simplicity and the function's derivative which can be easily computed and does not suffer from the problem of vanishing gradient. Therefore, ReLU will be used as the activation function in the hidden layers. ReLU has a range of zero to infinity. For the inputs which are less than 0, the output is zero and the inputs which are greater than 0, the output is equal to the input.

$$a = \phi(z) = max(z, 0) \tag{2.2}$$

SoftMax

As the activation function of the output layer of the multi-class classification problem, *softmax* is the state of the art. Its output corresponds to confidence for each class of being the actual class for a given feature set. The standard softmax equation can be written as,

$$a_j = \sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.3}$$

where z_i is the output of i^{th} unit of the output layer, a_i is the output of activation function for the i^{th} unit of the output layer, and K is the number of units in the output layer.

This function will be used as the activation function of the output layer.

2.4.2 Loss Function

At the training phase, to evaluate the effectiveness of training a loss function or cost function is used. It is a measurement of the distance between the predicted output and the actual output. The less the output of the loss function, the better the effectiveness of training. The goal of the neural network is to minimize the loss function. However, if the loss function is not convex shaped, the neural network may stick to a local optimum. *Categorical cross-entropy* is a convex shaped loss function for the multi-class classification problem, which can be defined as

$$L_i = \sum_j t_{i,j} log(p_{i,j}) \tag{2.4}$$

where t are the targets, i denotes the a feature vector, j denotes the class and $p_{i,j}$ is the probability of i belongs to class j.

2.4.3 Optimizer

To improve or optimize the performance of the neural network, that is minimizing the cost function, an optimizer is used.

Stochastic Gradient Descent

Stochastic Gradient Descent(SDG)[16] is an optimizer which iteratively minimizes the loss function. Instead of as a single group, or training set order, samples are selected randomly in this process. That's why the method is named stochastic.

The basic idea behind SGD is to compute an exponentially weighted average of the gradients of the loss, and then use that average gradient to update the weights of the connection between units from different layers. In figure 2.4, the algorithm is shown.

```
Require: Learning rate $\eta$.
Require: Initial parameter $\theta$.
while Stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{x^{(1)}, \dots, x^{(m)}\}$.
    Set $g = 0$
    for $i = 1$ to $m$ do
        Compute gradient estimate:
            $g \leftarrow g + \nabla_\theta L(f(x^{(i)};\theta)), y^{(i)};\theta)$.
    end for
    Apply update: $\theta \leftarrow \theta - \eta g$
end while
```

Figure 2.4: Stochastic gradient descent algorithm

Adaptive Moment Estimation

Adaptive moment estimation (Adam) is an algorithm for first-order gradient-based optimization of stochastic objective functions. It is based on adaptive estimates of lower-order moments. From estimates of first and second moments of the gradients, This method computes individual adaptive learning rates for different parameters. Kingma[17] showed that *Adam* outperforms all the other optimizers currently available. Figure 2.5 shows how fast Adam optimizes the model with iteration than other optimizers. The algorithm of *Adam* is shown in figure 2.6.

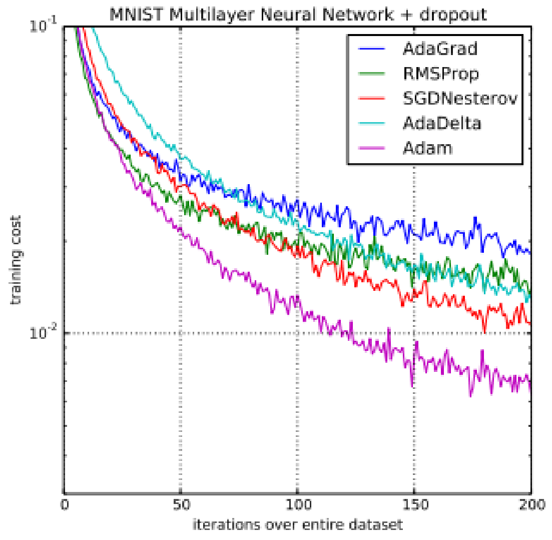
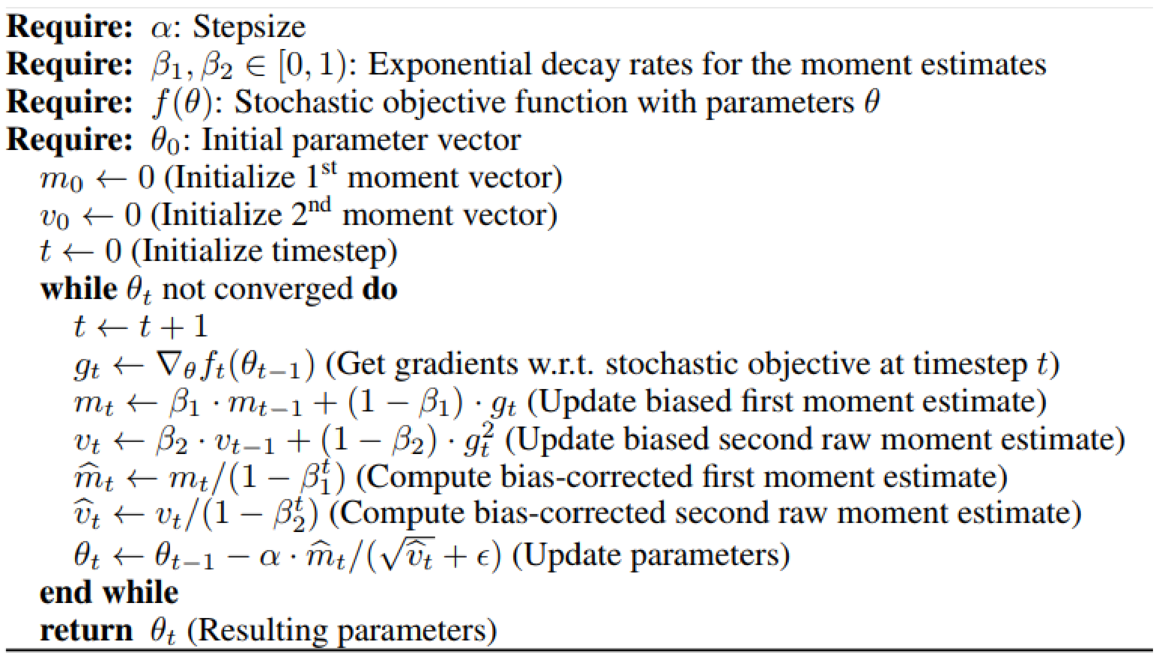


Figure 2.5: Comparison of Adam with other optimizers

2.4.4 Overfitting

The goal of a machine learning algorithm is to generate a model that will be generic to all feature sets in the problem domain. However, a finite number of data are used during the training of a model. As a result, it is quite common that a model performs very well for the data used in the training phase, but inhibits poor performance for new data. Such a situation is known as overfitting. As the name says, a situation, where the model is overfitted to the training data, is known as overfitting. In this



```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 2.6: Adam algorithm

case, the model has a high bias towards the training data. Careful considerations are needed to be made while training the data to prevent the model from having a high bias.

Regularization

As mentioned above, a machine learning algorithm performs a non-linear transformation from the input features to the predicted output. If the transformation becomes too non-linear, it loses the generosity and fits too much on the training data. As a result, it performs poor for newer data. Thus, necessary actions should be taken that prevents the model from being too much non-linear. Regularization does this job. Figure 2.7 shows a pictorial illustration of how regularization prevents the model from being overfitted.

Srivastava[18] proposed a relatively new regularization technique dropout. He showed that a neural network with dropout outperforms a neural network without

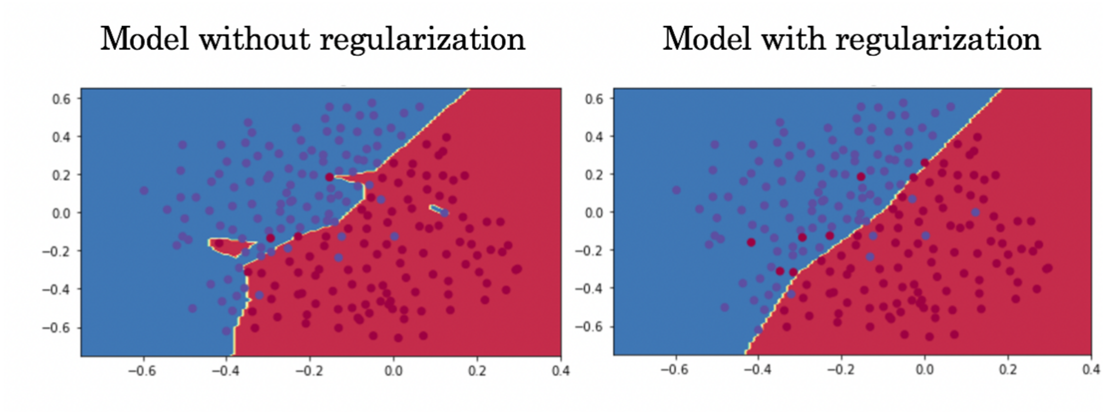


Figure 2.7: Pictorial illustration of how regularization prevents overfitting

dropout. The idea behind dropout regularization is shutting down a number of units from a particular layer randomly in every iteration. The units are to be shut down are different in different iterations. As different units of a layer are activated by different features, shutting a unit prevents overfitting for that particular feature. Finally, as a whole, the neural network is prevented from being overfitted.

2.5 Random Forest

Decision trees are used widely as classifier because of their high execution speed. But, the trees are designed to perfectly fit all samples in the training set which causes overfitting. Tim[19] introduced a method of constructing tree-based classifier whose capacity can be extended in order to increase the accuracy of both training set and test set.

Random forest is an ensemble learning method where each classifier in the ensemble is a decision tree classifier. This collection of classifiers is called a forest. During classification, each of the decision trees gives their votes and the result is based on the majority of votes. Figure 2.8 gives an overview of *random forest* ensembling method.

The set of attributes for a particular tree in the forest is randomly selected. The size of attribute subset is $\log_2 N + 1$ where N is the size of the attribute set.

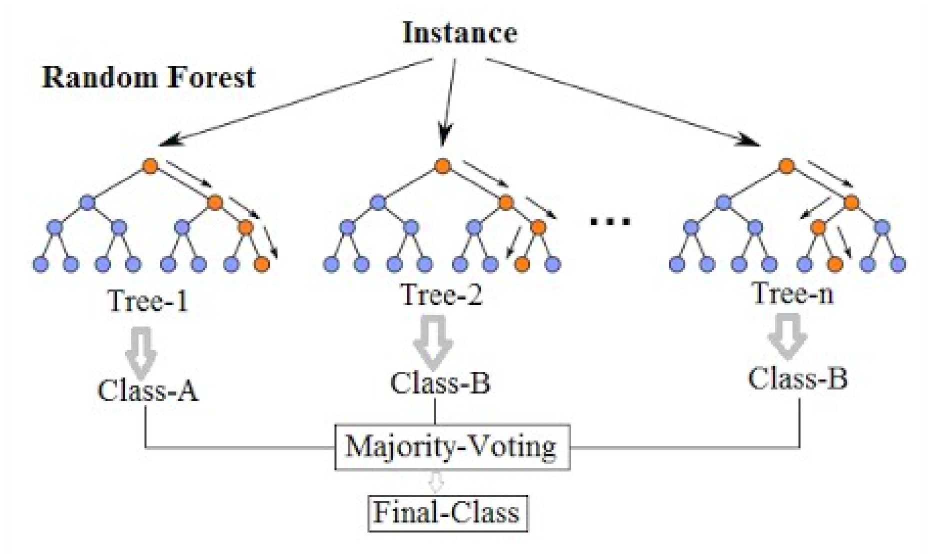


Figure 2.8: Overview of random forests

To construct the decision trees, *Classification and Regression Tree(CART)*[20] and *C4.5*[21] algorithms are used. Caruana[12] showed that in general for classification problems, random forest performs most consistently in all dimensions.

2.6 Support Vector Machine

Support vector machine(SVM) is a machine learning classifier which classifies data into different classes using a linear optimal separating hyperplane. This hyperplane is also known as a decision boundary.

SVM finds this hyperplane or decision boundary using support vectors and margins. Training points that are nearest to the separating function are known as the support vectors. The width that a decision boundary can have before hitting a support vector is known as margin. To classify the data, many hyperplanes can be produced. The optimal one maximizes the margins of the training data. This means it stays as far as possible from the support vectors of all classes.

Figure 2.9 shows an overview of SVM for linearly separable data. If the dataset is not in linearly separable feature space, then it is mapped into a higher-dimension

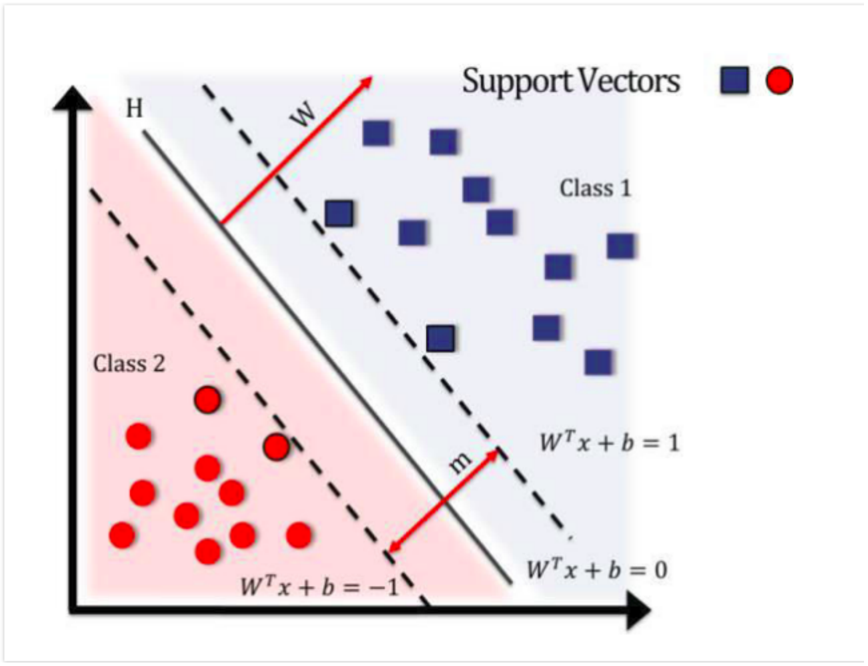


Figure 2.9: Overview of support vector machine

feature space using a kernel function where the dataset is linearly separable. In our thesis, we have chosen the *radial basis function(RBF)* kernel as our kernel function. Vert[22] defined The RBF kernel on two samples x and x', represented as feature vectors in some input space, as

$$K(x, x') = exp(-\frac{||x - x'||^2}{2\sigma^2}) \tag{2.5}$$

where $(||x - x'||)^2$ is the *squared euclidean distance* between two feature vectors. σ is a free parameter. According to Wang[23], The neural network minimizes the empirical training error. As a result, there exists a number of weak points of the network. The existence of many local minima and how to choose the number of hidden units are some of them. On the contrast, the SVM aims at minimizing an upper bound of the generalization error through maximizing the margin between the separating hyperplane and the data.

2.7 Ensembling Methods

By combining several methods, ensembling method helps to improve the results of machine learning. An ensemble is often more accurate than any of the single classifiers in the ensemble. According to Maclin[24], an ensemble consists of a set of individually trained classifiers whose predictions are combined, while classifying instances, by the ensemble method. These meta-algorithm combines several machine learning techniques into one predictive model. It tries to

- Decrease variance(Bagging)
- Decrease bias(Boosting)
- Improve prediction(Stacking)

In our thesis, we have used stacking in order to improve our prediction performance.

2.7.1 Stacking Ensemble Method

Aggarwal[25] stated, *stacking* is an ensemble learning method which combines multiple classifications using a meta-classifier.

In figure 2.10 we can see that the complete training set is used to train the individual classification models. Then based on the output, which is meta-features of the individual classification models in the ensemble, the meta-classifier are fitted. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble. The algorithm of stacking is shown in figure 2.11.

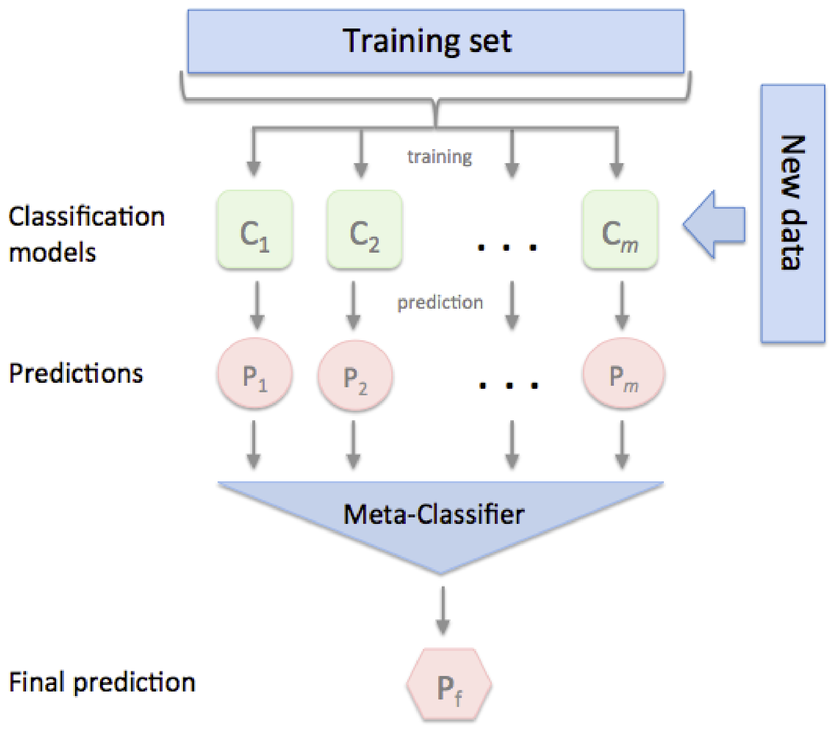


Figure 2.10: Overview of stacking

```
Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)
Output: An ensemble classifier $H$
 1: Step 1: Learn first-level classifiers
 2: for $t \leftarrow 1$ to $T$ do
 3:     Learn a base classifier $h_t$ based on $\mathcal{D}$
 4: end for
 5: Step 2: Construct new data sets from $\mathcal{D}$
 6: for $i \leftarrow 1$ to $m$ do
 7:     Construct a new data set that contains $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_T(\mathbf{x}_i)\}$
 8: end for
 9: Step 3: Learn a second-level classifier
10: Learn a new classifier $h'$ based on the newly constructed data set
11: return $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$
```

Figure 2.11: Stacking algorithm

Chapter 3

Related Works

Numerous works are available on source code author identification. They used a variety of features and classifiers. However, very few of them use machine learning techniques to identify the author of source codes. In the following sections, the background studies regarding this topic and related works are discussed.

According to Ďuračík[26], there are several approaches to identify the author of source code that can be divided into three levels. The first one is text-based and uses plain text as an input. The second level is token or metric based. The top level is model-based and uses models to represent source code.

3.1 Text Based Approaches

The first approach, which treats source code as plain text, is a form of natural language processing. This approach cannot make use of the programmatic structure of source code.

Frantzeskou et al.[1] proposed a technique called Source Code Author Profiles (SCAP) for author identification. They generated byte level n-gram author profile. To identify the author of source code, it's calculated profile is compared with previously calculated author profiles. They identified the author with the least dissimilar profile as the author of the source code. Burrows[27] mentioned, the SCAP method truncates the author profiles that are greater than the maximum profile length. This causes a bias towards the truncated profiles. To overcome the problem, the maximum profile length needs to be set to a large number that prohibits profile truncating. Therefore, a statistical analysis of the author profiles is required.

Burrows et al.[28] proposed an approach using information retrieval. They generated n-gram tokens from the source codes and indexed them in a search engine. Then, any source code can be queried for its author. In querying process, n-gram token of queried source code is generated and matched with author profiles. As query-result, a ranking list of all authors will be returned and the true author will top the ranking. This approach could identify the author of source code with 67% accuracy.

3.2 Metric Based Approaces

Frantzeskou[1] pointed out that metric-based author identification is divided into two parts. The first part is extracting the code metrics that represent the author's style and the second part is using those metrics to generate a model that is capable of labeling a source code by corresponding author name. However, the main disadvantage of this traditional approach is a vast amount of time is required to gather all possible metrics and examine to choose only the metrics that are responsible for differing the authors' style. This causes less attention to improving the efficiency and effectiveness of the proposed model.

Lange and Spiros[8] proposed a source code author identification technique using code metrics histogram and genetic algorithm. They assumed that the code metrics histogram should vary from author to author as of their coding style. From a number of source code metrics, an optimum set was selected using genetic algorithms. Then those metrics were used as input for the nearest neighbor classifier. This system was capable of identifying the author of the source codes with 55% accuracy. According to Yang[6], some of the features of this paper are unbounded. For example, the intendation category.

Shevertalov el al.[11] proposed a technique based on genetic algorithm. Firstly, the metrics are extracted from the source code. Then, they made a histogram from the source code metrics. This categorized histogram is sampled using the genetic algorithm. Finally, the author profile is produced using the categorized histogram samples. They tested the system with file input and project input. For files, they achieved 54% accuracy and for projects, they achieved 75% accuracy. Yang[6] mentioned that the details of the final feature set are not mentioned in this paper. So, the feature set is non-reproducible.

Bandara and Wijayarathna[15] used the deep neural network for source code author identification. Firstly, they converted the source codes into tokens or metrics. Their chosen tokens are identical to Lange et al.[8]. Then the tokens were fed to a deep neural network. Their deep neural network consisted of three restricted Boltzmann machine layers and one output layer. They achieved 93% accuracy.

Zhang et al.[9] used Support Vector Machine(SVM) to identify the author of source code. They categorized their feature into four groups namely – programming layout feature, programming style feature, programming structure feature and programming logic feature. They solved the problem in the domain of the multi-class classification problem and used sequential minimal optimization (SMO) as the classifier for SVM. Their datasets consisted of source codes from github.com and planetsourcecode.com and achieved 98% and 80% accuracy respectively.

Chapter 4

Author Identification of Source Codes Written by Multiple Authors

Our designed author identification approach is composed of four phases. Firstly, source code metrics will be extracted from the source codes in the training set. The extracted metric-values are then converted to feature vectors. Secondly, these feature vectors are fed to five individual base classifiers along with corresponding class-labels to train the base classifiers the author signatures. In the case of open source contribution, class-label means the owner of the source code and in case of a group of authors, the whole group is considered as the class label. By author signature, the coding style of a particular class-label is meant. Caruana[12] showed that in general for classification problem random forest, deep neural network, decision tree and support vector machine are the top four algorithms. Hence our chosen classifiers are the deep neural network(DNN), random forest with CART decision trees[20], random forest with C4.5 decision trees[21], C-support vector machine and ν-support vector machine. Each of the random forests has 100 decision trees built with the corresponding algorithm. Thirdly, each of the classifiers outputs the posterior class-probability according to their predictions. These outputs are called meta-features. Meta-features are used as the input for a meta-classifier. Then the meta-classifier is trained based on the meta-features and output. This approach is known as stacking ensemble. Another deep neural network is used as the meta-classifier. Figure 4.1 shows a block diagram of the architecture of the stacking ensemble method we have designed. Finally, to

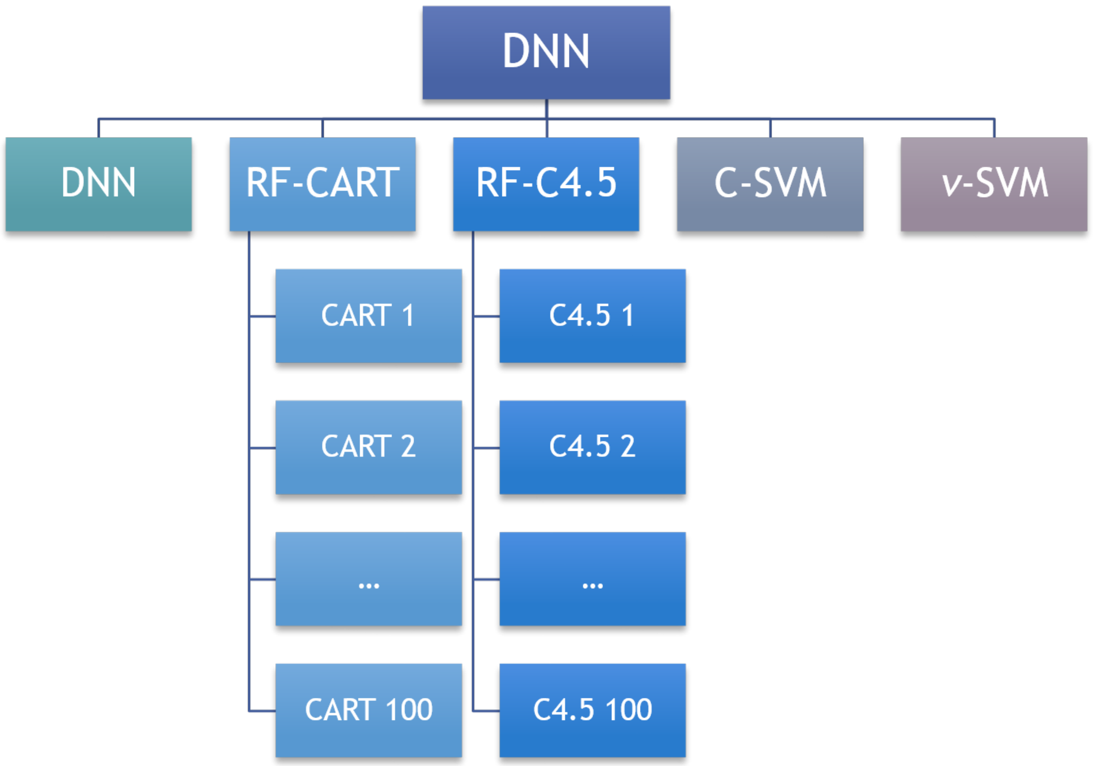


Figure 4.1: Block diagram of the architecture of the stacking ensemble method

identify the author of a new source code, that is from the test set, the same metrics are extracted from the test source code. These extracted metrics, after converting into feature vectors, are fed to the meta-classifier via the base classifiers. Using the experience from the training, the meta-classifier along with the base classifiers predicts the class labels of the test source codes. Figure 4.2 shows the block diagram of the proposed approach for author identification of source codes written by multiple authors.

In the following sections, the building blocks of the author identification approach

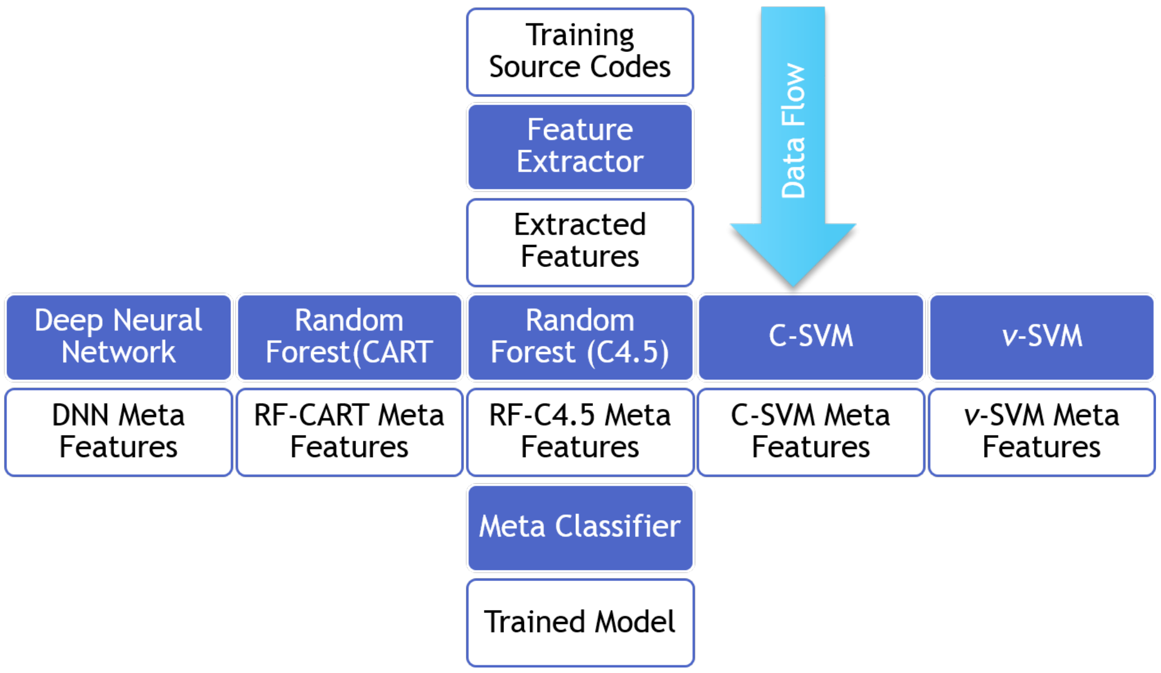


Figure 4.2: Block diagram of proposed author identification approach

are described.

4.1 Dataset

Some careful considerations are needed while choosing the dataset. Data must be collected from a diverse population of programmers and should provide enough information about the authors so that a clear distinction can be computed from author to author and valid comparison of their programming style can be made. In addition, the dataset must be close to real-world data as well as open for academic study[8].

In our study, we generated our dataset based on open source codes from github.com. All the source codes have a permissive license like MIT or BSD. The dataset contains 6063 python source codes from 8 authors/author groups which are considered as individual classes. Each source code contains roughly 226 lines on average. Source codes of each author are roughly split into 2:1 ratio to make the training and testing set. The training set contains 4034 files and the test set contains 2039 source codes.

Each class label consists of authors and contributors. By author, we mean the true owner of the projects. This can be a single author or a group of authors. By contributors, we mean a group of people who are not the owner of the project but willingly contributes to the project by writing or editing a segment of it. In general, the number of lines of code written or edited by authors is far greater than the lines of code written or edited by contributors. On the contrast, contributors generally out-number the authors. The number of authors and the number of contributors per class-label is listed in table 4.1.

Table 4.1: Number of authors and contributors for each class

Class Label	**Number of Authors**	**Number of Contributors**
Azure	3	136
GoogleCloudPlatform	33	820
StackStorm	2	147
dimagi	2	101
enthought	9	224
fp7-ofelia	1	4
freenas	2	126
sympy	2	712

4.2 Metric Extraction

Previously, Shevertalov, Lange, Bandara, and Zhang[8, 9, 11, 15] used source code metrics for author identification. From a set of probable code metrics, Lange selected the optimal set of code metrics using the genetic algorithm. Bandara used almost the same set of source code metrics. We used the same set of metrics for our author identification approach only except the access modifier metric. The access modifier feature is present only a limited number of programming languages and makes the whole system language dependent. Table 4.2 shows the set of metrics to be used and corresponding descriptions.

Table 4.2: Set of code metrics and descriptions

Metric Name	**Metric Description**
Line Length Calculator	This metric measures the number of characters in one source code line.
Line Words Calculator	This metric measures the number of words in one source code line.
Comments Frequency Calculator	This metric calculates the relative frequency of line comment, block comment and optionally doc-comment used by the programmers.
Identifiers Length Calculator	This metric calculates the length of each identifier of programs.
Inline Space-Tab Calculator	This metric calculates the whitespaces that occur on the interior areas of non-whitespace lines.
Trail Space-Tab Calculator	This metric measures the whitespace and tab occurrence at the end of each non-whitespace line.
Indent Space-Tab Calculator	This metric calculates the indentation whitespaces used at the beginning of each non-whitespace line.
Underscores Calculator	This metric measures the number of underscore characters used in identifiers.

The number of extracted metrics is not consistent from author to author. For example, say a source code has 200 lines. So it will have 200 entries for line length metric. On the other hand, another source code has 500 lines. So it will have 500 entries for line length metric. However, the neural network requires to have a fixed input size. So, we converted the extracted source code metrics to a form that can be fed to a neural network. At first, we performed outlier analysis for each of the metrics and fed them to the base classifiers. We determined the IQR for each of the code metrics. Metrics outside of range $(Q_1 - 1.5IQR, Q_3 + 1.5IQR)$ are considered as an outlier. During the outlier analysis phase, we roughly selected the following ranges for the source code metrics.

- **Line Length**: 0 to 118
- **Line Word**: 0 to 18
- **Identifier Length**: 0 to 18
- **Underscore Count**: 0
- **Indentation Space-Tab**: 0 to 18
- **Inline Space-Tab**: 0 to 8
- **Trailing Space-Tab**: 0

After that, for each source code, we counted the number of all possible metric values. For example, for line word metric, we counted how many lines with 0 words, how many lines with 1 word, how many lines with 2 words, and this way till lines with 18 words and how many lines with more than 18 words for each source code. Especially, for comments, we just counted the number of line comments and the number of block comments. This way line length generates 120 values, line word generated 20 values, identifier length generated 20 values, underscore count generates 2 values(0 and more than 0), indentation space-tab generates 20 values, inline space-tab generates 10 values, trailing space tab generates 2 values(0 and more than 0), and two values for line comment and block comment. Finally, it results a feature vector of size 196.

4.3 Base Classifiers

There are a total of five base classifiers in our author identification system. They are – deep neural network, random forest based on CART, random forest based on C4.5, C-support vector machine and ν-support vector machine. Each of the base classifiers is described below.

4.3.1 Deep Neural Network

The DNN model used as the base classifier consists of 14 layers. Data are fed to the DNN as batches of 32 entries. The input layer transforms each batch into a standard normal distribution of which standard deviation is 1 and mean is 0. Then there are eight densely connected layers, followed by a dropout layer, a densely connected layer, a dropout layer, a densely connected layer and finally the output layer. The dropout layers ensure that the classifier is not too biased towards training data. The output layer has the same number of units as the number of classes. Figure 4.3 depicts the layers and corresponding input-output sizes of DNN base classifier.

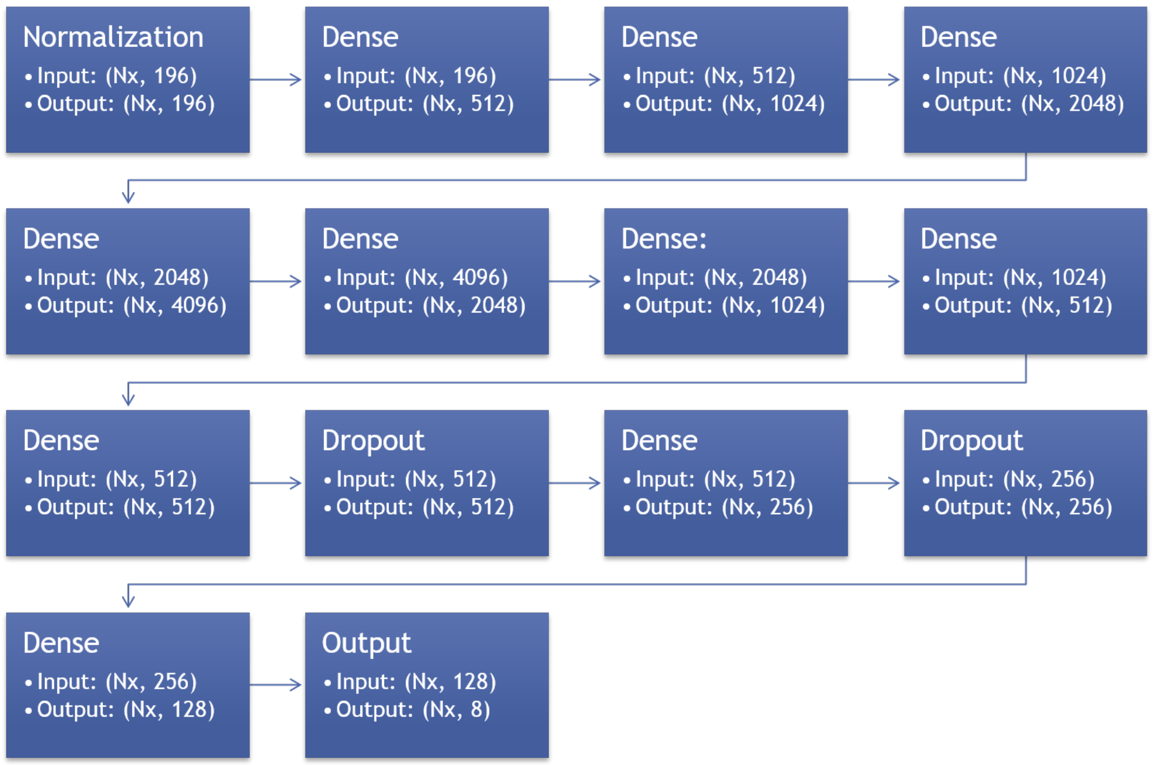


Figure 4.3: Block diagram of the neural network used as base classifier

In the densely connected layers, $ReLU$ activation function and in the output layer *softmax* activation function are used. *Categorical cross-entropy* is chosen as the loss function. *Adam* optimizer is used to optimize the network. The neural network outputs posterior probability for each class.

4.3.2 Random Forest

The second base classifier is a random forest with one hundred decision trees. Classification and Regression Tree(CART)[20] algorithm is used to build the trees which select the split node based on gini impurity.

The third base classifier is another random forest with one hundred decision trees. Unlike the second base classifier, decision trees in the third base classifier are built with the C4.5[21] algorithm. This algorithm chooses the split node based on the entropy ratio.

The output of both the classifiers is the average of the predictions from the corresponding decision trees. This average can be interpreted as the posterior probability for each class.

4.3.3 Support Vector Machine

The fourth base classifier is a C-support vector classifier. It is a support vector machine where C is a penalty parameter for the error term.

The fifth base classifier is a ν-support vector classifier. It is a support vector machine where ν is the upper bound of training error and the lower bound of the number of support vectors.

The output of both the support vector machines is the posterior probability for each class.

4.4 Meta Classifiers

We used another deep neural network as the meta-classifier. The outputs of the base classifiers(meta-features) are fed to the meta-classifier to learn the mapping from the meta-features to the actual output.

The building blocks of the deep neural network used as meta-classifier are depicted in figure 4.4.

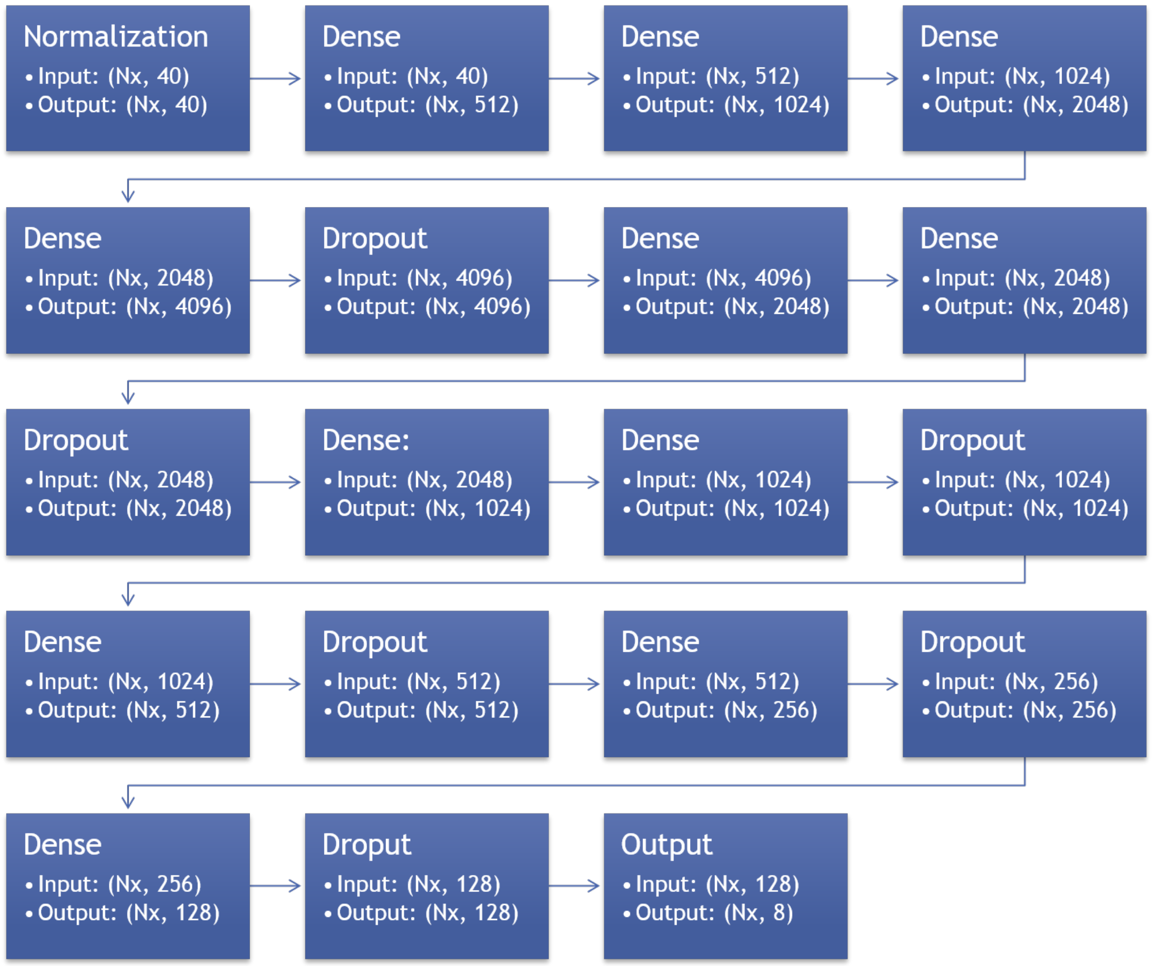


Figure 4.4: Block diagram of the neural network used as meta classifier

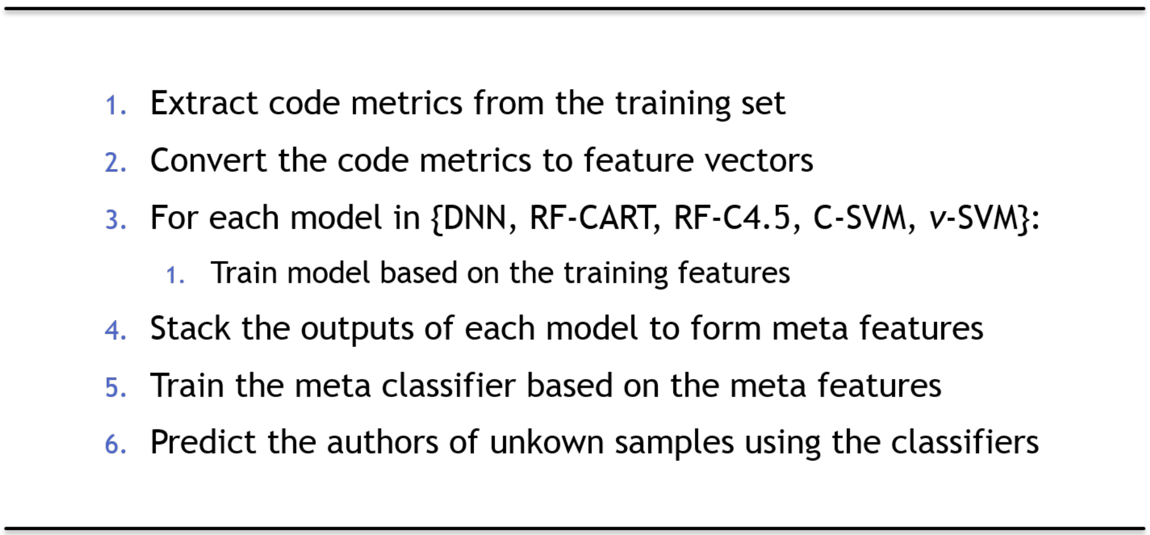
The neural network consists of 19 layers. Data are fed to the meta-classifier as batches of 32 feature vector each. The first layer of the network transforms the data into standard normal distribution. This layer is followed by eight densely connected layers, a dropout layer, two densely connected layers, a dropout layer, a densely connected layer, a dropout layer, a densely connected layer, a dropout layer, a densely connected layer, a dropout layer, and finally the output layer. The output from this output layer is the final output of our author identification system for source code written by multiple authors.

The activation functions of the network are *ReLU* for densely connected layers and *softmax* for the output layer. The loss function used in the meta-classifier is *categorical cross-entropy*. *Stochastic Gradient Descent(SGD)* is used as the optimizer of the meta-classifier.

4.5 Training

We implemented our author identification system for source code written by multiple authors in multi-class classification category. Here, a unique list of authors(or groups of authors) of the source codes in the training set are treated as classes. The author identification system will output its confidence for each class of being the actual class of given source code. The actual author is expected to have the highest confidence.

The training phase of our system is divided into three phases – feature extraction from the source codes, training the base classifiers and training the meta-classifier. Figure 4.5 shows the steps followed in our author identification system for source code written by multiple authors.



1. Extract code metrics from the training set
2. Convert the code metrics to feature vectors
3. For each model in {DNN, RF-CART, RF-C4.5, C-SVM, *v*-SVM}:
    1. Train model based on the training features
4. Stack the outputs of each model to form meta features
5. Train the meta classifier based on the meta features
6. Predict the authors of unkown samples using the classifiers

Figure 4.5: Steps for training the stacking ensemble system

First of all, the source code metrics mentioned in table 4.2 are extracted from source codes. Then they are converted to feature vector as mentioned in section 4.2. These feature vectors are fed to each of the base classifiers as input.

The base classifiers run according to their own learning algorithm to learn to identify the writing style of each class. During this training phase, several configurations of each of the base classifiers, specially DNN are used to find out which configuration works the best for the training set.

After completing the training of each of the base models, the posterior probability for each input in the training set is generated. This produces a $5 * |classes|$ sized feature vector for each of the input feature vectors where $|classes|$ is the number of classes. These feature vectors are known as meta-features. Meta features are fed to the meta-classifier along with the class labels through which the meta-classifier learns to predict the actual class from the meta-features.

Chapter 5

Experimental Results

5.1 Experimental Setup

The experimental setup consists of the toolset we have used to implement our proposed method and the values we have fixed for the parameters of the classifiers.

5.1.1 Toolset

While implementing our author identification system for source code written by multiple contributors, we used Keras[29] as the framework for deep neural networks and Scikit Learn[30] as the library for general purpose machine learning. For data preprocessing and visualization, we used numpy[31] and pandas[32] library.

First of all, we made a feature extractor that extracts the features mentioned in table 4.2 from the source codes. Then we converted the code metrics to feature vectors. Among the base classifiers, random forest with CART decision trees, random forest with C4.5 decision trees, C-support vector machine and ν-support vector machine are built with *Scikit Learn* library. The deep neural network as the base classifier and the deep neural network as the meta-classifier are built with *Keras*.

5.1.2 Parameters

During the experiment, we found that for both the random forests, a hundred trees are sufficient to converge to the highest accuracy. For C-support vector machine, the parameter C is a penalty for the error term. For ν-support vector machine, the parameter ν is an upper bound to the training error and lower bound to the number of support vectors. We found, for particularly our problem, the optimal value for C and ν are 1.0 and 0.15 respectively. For the deep neural network as the base classifier, the learning rate is 0.01. For the optimizer of this classifier, which is Adam, the parameters β_1 is 0.9 and β_2 is 0.999. For the deep neural network as the meta-classifier, the learning rate is 0.001. For the optimizer of this classifier, which is SGD, the parameter *momentum* is 0.

5.2 Metrics

To evaluate our method for author identification of source codes written by multiple authors, we used two metrics. These are *accuracy* and *f1-score*.

5.2.1 Accuracy

Accuracy is the ratio between the number of correctly identified samples and the number of total samples. Mathematically,

$$accuracy = \frac{|correctly\ identified\ samples|}{|total\ samples|}$$

where, $|X|$ is the number of items in X set.

5.2.2 F1-Score

F1-score is the harmonic mean of *precision* and *recall*. *Precision* is the ratio between the number of correctly identified samples for a particular class-label and the number of total samples identified as of that class-label. *Recall* is the ratio between the number of correctly identified samples for a particular class-label and the number of total samples actually belongs that class-label.

As the definitions suggest, *precision* and *recall* are individually calculated for each of the class-labels. Then these scores can be aggregated in several ways to compute the final *f1-score*. In our work, we used *micro averaging* to compute the *f1-score*. In *micro averaging*, the total number of correctly identified samples and the total number of incorrectly identified samples are computed globally. Mathematically,

$$precision = \frac{|correctly\ identified\ samples\ of\ class\ A|}{|total\ samples\ identified\ as\ of\ class\ A|}$$

$$recall = \frac{|correctly\ identified\ samples\ of\ class\ A|}{|total\ samples\ of\ class\ A|}$$

$$f1\text{-}score = \left(\frac{2}{precision^{-1} + recall^{-1}}\right)^{-1}$$

where, $|X|$ is the number of items in X set.

5.3 Results of The Base Classifiers

Table 5.1 contains the accuracies for the five base models of our stacking ensemble method. From the table, we can see that the highest accuracy is achieved for random forest classifiers. The accuracies of both the random forests are 83%. However, their testing predictions match in 91.67% cases. Similarly, for support vector machines, the prediction-match is 83.91% although the accuracy was 79% for both the classifiers.

5.4 Results of The Meta Classifier

After training the meta-classifier by the meta-features, we achieved 87% accuracy. The f1-score of our work is 0.86.

Table 5.1: Accuracy of the base classifiers

Classifier Name	**Accuracy**
Deep Neural Network	82%
CART Based Random Forest	83%
Random Forest	83%
C-Support Vector Machine	79%
ν-Support Vector Machine	79%

Table 5.2 shows the confusion matrix of our work. A confusion matrix is a table describing the performance of a supervised learning algorithm. Here the row label denotes the actual class-label and column label denotes the predicted class-label. In the corresponding cell, the number of samples with such actual class-label and predicted class-label is shown.

Table 5.2: The confusion matrix of for the testing dataset

	Azure	**Google-Cloud-Platform**	**Stack-Storm**	**dimagi**	**en-thought**	**fp7-ofelia**	**freenas**	**sympy**
Azure	171	2	1	0	2	0	0	0
Google-Cloud-Platform	5	253	7	3	8	2	0	2
Stack-Storm	0	2	180	33	9	2	1	0
dimagi	1	2	2	383	17	17	3	8
en-thought	2	4	2	29	339	7	0	11
fp7-ofelia	1	3	3	23	16	141	0	3
freenas	0	1	1	11	5	2	162	3
sympy	1	1	0	8	11	3	3	117

Table 5.3 shows a comparison between our work and other related works. The comparison includes the type of features, language independence, the capability of handling multiple authorship, number of classes and the total number of source codes used in training and testing. From the table, we can see that the recent works on source code author identification tend to use code metrics rather than n-gram as we did. Our chosen set of metrics is compact and still able to achieve a satisfactory accuracy. Alongside a number of works suffer from choosing a set of metrics that are not language independent.

Table 5.3: Comparison among the methods for source code author identification

Method Name	**Features**	**Language independent features**	**Multiple author**	**Number of classes**	**Total Source code**	**Accuracy**
Information retrieval approach[28]	Character level n-gram	Yes	No	100	1640	67%
Code metric histogram[8]	7 code metrics	Yes	No	20	4068	55%
Genetic algorithm[11]	4 code metrics	Yes	No	20	N\A	75%
Deep neural network[15]	9 code metrics	No	No	10, 10, 8, 5, 9	1644, 780, 475, 131, 520	93%, 93%, 93%, 78%, 89%
Support vector machine[9]	46 code metrics	No	No	8, 53	8000, 502	98%, 80%
Stacking ensemble method	8 code metrics	Yes	Yes	8 (group of authors)	6063	87%

Chapter 6

Conclusion

In this thesis, we designed a new approach for identifying the author of a source code where the number of authors of the source codes is more than one. We used stacking ensemble method for our author identification task. Stacking ensemble is an approach where a number of heterogeneous base classifiers are stacked together to form a new classifier. In general, stacking ensemble method performs better than any of its base classifiers. The main challenge of this method is to select the base estimators from a large number of possible combinations. Again, as it is required to train several classifiers, each classifier needs to be fine tuned individually to produce a good final result. On the other hand, the problem of identifying the authorship of source codes is harder when the number of authors is more than one. Because then the writing style of the source code is inconsistent from segment to segment.

We have designed a stacking ensemble classifier that consists of five base classifiers and a meta-classifier. Our designed classifier is able to classify the authorship of source codes written by multiple authors with 87% accuracy. Alongside, we have chosen a relatively small set of code metrics that are relatively easy to compute and language independent as well.

6.1 Future Direction

Although our stacking ensemble method achieved a satisfactory accuracy, this still can be improved. Although our code metrics are language independent, we only tested with python source codes. Future works may test on other languages and check how the set of metrics works for other languages. Other sets of metrics can also be examined to see how they contribute to the writing style of source codes.

Bibliography

[1] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas, "Source code author identification based on n-gram author profiles," in *Artificial Intelligence Applications and Innovations* (I. Maglogiannis, K. Karpouzis, and M. Bramer, eds.), (Boston, MA), pp. 508–515, Springer US, 2006.

[2] J. Kothari, M. Shevertalov, E. Stehle, and S. Mancoridis, "A probabilistic approach to source code authorship identification," in *Proceedings of International Conference on Information Technology: New Generations*, IEEE, 2007.

[3] J. Zobel, "Uni cheats racket: A case study in plagiarism investigation," in *Proceedings of the sixth conference on Australasian computing education*, vol. 30, (Australia, Australia), pp. 357–365, Australian Computer Society, Inc. Darlinghurst, 2004.

[4] G. Frantzeskou, E. Stamatatos, and S. Gritzalis, "Supporting the cybercrime investigation process: Effective discrimination of source code authors based on byte-level information," in *E-business and Telecommunication Networks* (J. Filipe, H. Coelhas, and M. Saramago, eds.), (Berlin, Heidelberg), pp. 163–173, Springer Berlin Heidelberg, 2007.

[5] A. Gray, P. Sallis, and S. MacDonell, "Identified: A dictionary-based system for extracting source code metrics for software forensics," in *Procceedings of SE: EI&P*, (Washington, DC), pp. 252–259, IEEE Computer Society Press, 1998.

[6] X. Yang, G. Xu, Q. Li, Y. Guo, and M. Zhang, "Authorship attribution of source code by using back propagation neural network based on particle swarm optimization," *PLOS ONE*, vol. 12, pp. 1–18, 11 2017.

[7] M. F. Tennyson and F. J. Mitropoulos, "A bayesian ensemble classifier for source code authorship attribution," in *SISAP* (A. M. T. et al., ed.), vol. 8821 of *LNCS*, (Switzerland), p. 265–276, Springer International Publishing, 2014.

[8] R. C. Lange and S. Mancoridis, "Using code metric histograms and genetic algorithms to perform author identification for software forensics," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, (New York, NY, USA), pp. 2082–2089, ACM, 2007.

[9] C. Zhang, S. Wang, J. Wu, and Z. Niu, "Authorship identification of source codes," in *Proceedings of Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data* (C. L., J. C., S. C., Y. X., and L. X., eds.), vol. 10366 of *Lecture Notes in Computer Science*, (Cham, Switzerland), pp. 282–296, Springer, 2017.

[10] O. Mirza and M. Joy, "Style analysis for source code plagiarism detection," in *Proceedings of International Conference on Plagiarism across Europe and Beyond*, (Brno, Czech Republic), pp. 53—-61, 2015.

[11] M. Shevertalov, J. Kothari, E. Stehle, and S. Mancoridis, "On the use of discretized source code metrics for author identification," in *Proceedings of 1st International Symposium on Search Based Software Engineering* (M. D. Penta and S. Poulding, eds.), (Cumberland Lodge, Windsor, UK), pp. 69–78, IEEE Computer Society, 2009.

[12] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th International*

Conference on Machine Learning, ICML '08, (New York, NY, USA), pp. 96–103, ACM, 2008.

[13] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210 – 229, 1959.

[14] T. Mitchell, *Machine Learning.* McGraw Hill, 1997. ISBN: 0-07-042807-7.

[15] U. Bandara and G. Wijayarathna, "Deep neural networks for source code author identification," in *Neural Information Processing* (M. Lee, A. Hirose, Z.-G. Hou, and R. M. Kil, eds.), (Berlin, Heidelberg), pp. 368–375, Springer Berlin Heidelberg, 2013.

[16] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 09 1951.

[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Procceedings of 3rd International Conference for Learning Representations*, (San Diego), 2015.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," in *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[19] T. K. Ho, "Random decision forest," in *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, (Montreal, QC), pp. 278–282, 1995.

[20] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and Regression Trees.* CRC Press, 1984.

[21] Q. JR, *C4.5: Programs for Machine Learning.* San Mateo: Morgan Kaufmann, 1993.

[22] J. Vert, K. Tsuda, and B. Sch ö lkopf, *A Primer on Kernel Methods.* Cambridge, MA, USA: MIT Press, 2004.

[23] J. Wang, Q. Chen, and Y. Chen, "Rbf kernel based support vector machine with universal approximation and its application," in *Proceedings of Advances in Neural Networks – ISNN 2004*, Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 512–517, Springer, 2004.

[24] R. Maclin and D. Opitz, "Popular ensemble methods: An empirical study," *Journal Of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[25] C. C. Aggarwal, *Data Classification: Algorithms and Applications.* Data Mining and Knowledge Discovery Series, CRC Press, 2015.

[26] M. Duracik, E. Krsak, and P. Hrkut, "Current trends in source code analysis, plagiarism detection and issues of analysis big datasets," in *TRANSCOM 2017: International scientific conference on sustainable, modern and safe transport*, vol. 192 of *Elsevier: Procedia Engineering*, pp. 136–141, 2017.

[27] S. Burrows, A. Uitdenbogerd, and A. Turpin, "Comparing techniques for authorship attribution of source code," *Software: Practice and Experience*, vol. 44, 01 2014.

[28] S. Burrows and S. Tahaghoghi, "Source code authorship attribution using n-grams," in *Proceedings of the Twelfth Australasian Document Computing Symposium* (A. T. Amanda Spink and M. Wu, eds.), pp. 32–40, School of Computer Science and Information Technology, RMIT University, 2007.

[29] F. c. o. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in P ython,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[31] O. Travis E, “A guide to numpy,” 2006.

[32] W. McKinney, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51–56, 2010.