Authorship Analysis: Source Code Identification

1st Jonas Fehlhauer
Hochschule Mittweida
Computer- und Biowissenschaften
09648, Mittweida
jfehlhau@hs-mittweida.de

2nd Vladislav Förster
Hochschule Mittweida
Computer- und Biowissenschaften
09648, Mittweida
vfoerst1@hs-mittweida.de

3rd Christina Gneuß
Hochschule Mittweida
Computer- und Biowissenschaften
09648, Mittweida
cgneuss1@hs-mittweida.de

4th Shirley Klier
Hochschule Mittweida
Computer- und Biowissenschaften
09648, Mittweida
sklier@hs-mittweida.de

5th Josefine Rothe
Hochschule Mittweida
Computer- und Biowissenschaften
09648, Mittweida
jrothe2@hs-mittweida.de

***Abstract*—Mit dem Fortschritt der Technik soll einem eigentlich das Leben erleichtert werden. Jedoch eröffnet es auch neue Gebiete für die Verbrechenswelt, was Probleme verursachen kann. Daher ist es zwingend erforderlich, dass solche Aktivitäten festgestellt, verfolgt und nachgewiesen werden können. Besonders die Verbesserung der dafür verwendeten Werkzeuge ist heutzutage ein bedeutender Wirtschaftszweig. Der Diebstahl geistigen Eigentums in Form von Texten ist bereits Jahrhunderte lang bekannt. Die Übertragung dessen auf Quellcode ist dagegen noch wenig verbreitet, obwohl keine App ohne einen solchen existieren würde. Unter anderem deshalb ist es inzwischen nicht nur wichtig Plagiate normaler Texte identifizieren zu können, sondern auch von Quelltexten. Source Code Identification, oder zu Deutsch Quellcode Identifikation, beschäftigt sich mit der Analyse verschiedener Charakteristiken eines Codes um dadurch den Autor benennen zu können. Dies kann in Fällen von Cyberattacken, Computerbetrug und vor allem zur Identifizierung von Plagiaten genutzt werden. In diesem Paper wird ein selbstentwickeltes Programm erläutert, welches einen Quellcode einem bestimmten Autor zuweisen kann. Dabei wird neben der Support Vector Machine (SVM) und den verwendeten Features, die zur Umsetzung verwendet wurden, auch auf Probleme eingegangen. Da jeder Programmierer andere lexikalische, syntaktische und das Layout betreffende Eigenheiten besitzt, zum Beispiel bedingt durch einen unterschiedlichen Wissensstand aber auch seinen eigenen Stil beim Programmieren, kann somit jeder durch seine individuelle Schreibweise durch genügend Unterschiede aus einer Gruppe von Personen authentifiziert werden.**

***Index Terms*—Support Vektor Machine, Source Code Identification, Feature**

I. Einleitung

Jedes technische Gerät benötigt ein Programm, das mit diesem interagiert um Funktionen wie das einfache Starten umsetzen zu können. Sie werden täglich genutzt und erleichtern uns den Alltag, doch die meisten wissen nicht, wie diese funktionieren. Das Fundament eines Programms ist ein sogenannter Quellcode beziehungsweise Quelltext. Er wird vom Programmierer in einer für Menschen leicht lesbaren Form geschrieben und erst durch eine Vorverarbeitung in Maschinencode umgewandelt, um im Anschluss ausgeführt werden zu können. Dies mag für Leute, die sich mit Computern nicht auskennen dennoch unverständlich sein. Das liegt daran, dass sie mithilfe einer Programmiersprache erstellt wird. Wie jede andere Sprache muss auch sie erst erlernt werden, bevor sie verstanden werden kann [vgl. G]. Durch die steigende Nutzung von Technik stieg auch die Nachfrage nach Programmierern. Daraus resultierend ist mittlerweile ein Konkurrenzkampf unter Programmierern entstanden. Wie bei nahezu jedem Wettbewerb gibt es natürlich auch hier Menschen, die sich versuchen das Leben einfacher zu machen und beispielsweise einen Quelltext aus dem Internet nehmen und diesen als ihren eigenen ausgeben. Um solche Betrügereien aufzudecken aber auch andere Herausforderungen, wie die Zurückverfolgung der Herkunft von Viren und ähnlichem, zu lösen wurde die Analyse von Texten bezüglich ihres Autors ausgeweitet auf den Bereich Quellcode, auch genannt Authorship Analysis - Source Code Identification. Eine Definition von Source Code Authorship Identification erklärt, dass die Aufgabe darin besteht, aus einer Menge P von Programmierern und durch ihre Quellcode-Beispiele festzustellen, welcher dieser Programmierer aus P ein Programm mit unbekanntem Autor geschrieben hat. [2, S. 285] Dafür werden in der Regel Feature verwendet die in verschiedenen Lösungsansätze unterschiedlich verarbeitet werden. Gleichbleibend ist aber, dass Feature immer darauf abzielen die einzigartige Stilometrie eines Autors zu bestimmen. In diesem Paper wird die Entwicklung eines Systems, das den richtigen Autor eines Quellcodes aus einer Liste von 1000 Autoren ermittelt dargelegt. Für die Bearbeitung stand ein Datensatz von je 75 Quellcodes eines jeden Programmierers zur Verfügung. Als Baseline wurde die Accuracy von TF-IDF KNN mit 0,62 ausgewählt. Dabei wurden die Quellcodes nach der TF-IDF-Methode mit 10K-Merkmalen vektorisiert und ein KNN-Klassifikator mit 25 Nachbarn, auf den aus TF-IDF extrahierten Darstellungen

Identify applicable funding agency here. If none, delete this.

erstellt. Es wird im Einzelnen ausführlich auf den verwendeten Lösungsansatz in Form einer SVM, den Entwicklungsprozess an sich aber auch auf Probleme währenddessen eingegangen. Außerdem wird die Datenlage inklusive aller genutzten Feature, die Methoden und die Ergebnisse dargelegt, sowie eine Literaturdiskussion durchgeführt.

II. Literaturdiskussion

Während der Suche nach ähnlicher Literatur zu Authorship Analysis für den Bereich Source Code, tauchten mehrere verschiedene Lösungsansätze auf. Beispielsweise Source Code Author Profiles (SCAP) welche auf byte-level n-gram Profilen basiert, um den Stil eines Autors zu repräsentieren [1], aber auch ein Convolutional Neural Network (CNN), das unter anderem word embedding modelling und Feature-lernende Techniken verwendet [12]. Allerdings zeigte sich, dass eine SVM als Lösungsansatz zwar durchaus für die Analyse literarischer Werke bereits verwendet wurde, jedoch nicht bezogen auf die Untersuchung eines Autors von Source Code. Dies bot einen Anreiz es für diesen Bereich zu testen. Auch wenn der Ansatz ein anderer war, wurde sich in der Literatur dennoch ebenfalls mit Feature Extraktion beschäftigt. Einige Feature die in anderen Papern gefunden wurden, konnten übertragen oder in ähnlicher Weise verwendet werden. Beispielsweise aus "Identifying the author of a program" von Krsul [vgl. 8] entstand das Feature Kommentarlänge wie in Tabelle 1 erklärt. Auch das Feature Character Count (Tabelle 1) tauchte in anderer Literatur similär auf [11]. Obwohl sich durch eine steigende Anzahl an Featuren eine Verbesserung feststellen lässt zeigt sich in anderen Arbeiten, dass auch wenige Feature bei gewissen Lösungsansätzen bereits eine ansehnliche Accuracy erreichen können. So konnte mit nur vier Featuren und einem Genetic Algorithm eine Genauigkeit von 75 Prozent erzielt werden. Allerdings muss erwähnt werden, dass die Anzahl an Autoren nur bei 20 lag und 750.000 Zeilen Java Source Code betrachtet wurden. Das bedeutet, dass es zwar durchaus bei einer geringen Anzahl an zu unterscheidenden Autoren bei genügend vorhandenem Code nützlich ist, aber bei steigender Autorenzahl vermutlich keine besonders hohe Accuracy mehr erreicht werden würde. Im Laufe der Recherche wurde dann festgestellt, dass die Identifikation des richtigen Autors aus einem Pool von 1000 Programmierern oder einer ähnlich hohen Anzahl nicht üblich ist. In nahezu allen Beiträgen betrug die Anzahl an zu unterscheidenden Autoren nicht einmal annähernd 100. Es ließ sich lediglich ein Paper mit 1600 Autoren finden. Bei diesem Ansatz wurden manuell gefertigte Feature von einem Abstract Syntax Tree (AST) verarbeitet und in einer Random Forest Classification angewandt [12, 14]. Mit einer Accuracy von 92,8 Prozent übertrifft es zwar den in diesem Paper verwendeten Ansatz, jedoch ist das bei Machine Learning und dem dafür benötigten Rechenaufwand zu erwarten. In einem Kapitel aus dem Buch "Big Data" von L. Chen et al. (ab Seite 282), wurde mit zwei verschiedenen Datensätzen gearbeitet. Der erste mit 53 Autoren bestand aus einem unbalancierten Datensatz mit insgesamt 502 Programmen, während beim anderen nur zwischen acht Autoren unterschieden werden musste, denen jedoch jeweils 1000 Quellcodes zugeordnet werden sollten. Es wurden hierbei sowohl Decision tree, als auch Random Forest und Sequential Minimal Optimization (SMO) verwendet. Die Ergebnisse in Form einer Identifikations-Accuracy zeigen unabhängig von der genutzten Methode, dass eine Unterscheidung weniger Autoren deutlich leichter und mit einer hohen Genauigkeit möglich ist. [2] Auch die unterschiedlich hohe Anzahl an Quellcodes spielt eine Rolle bei der Genauigkeit der Zuordnung, da bei einer höheren Anzahl an Objekten auch mehr zum Trainieren genutzt werden kann. Ein direkter Vergleich der Genauigkeit mit dem Datensatz in diesem Paper mit 1000 Autoren lässt sich daher nicht aufstellen. Allerdings zeigte sich während der fortschreitenden Entwicklung, dass auch die SVM, bei einer großen Anzahl von Autoren und Programmen und außerdem zunehmender Featureanzahl, eine beträchtliche Rechenkapazität benötigt. Daher ist es nur gelungen eine Accuracy für eine maximale Anzahl von 200 Autoren zu erstellen, denen jeweils höchstens 75 Quellcodes zugeordnet werden mussten. Es wurden daher mehrere Durchläufe für 50, 100 und 200 Autoren durchgeführt, um dennoch Verbesserungen vorzunehmen. Werden diese Werte mit denen in [2] verglichen lässt sich feststellen, dass die Accuracy der SVM für 100 Autoren mit etwa 0.75 niedriger ist. Jedoch hat sie sich im Vergleich zum Beginn des Projekts, wo die Accuracy für 100 Autoren bei 0.63 lag, deutlich gesteigert. Ein Vergleich mit Ergebnissen aus anderen Beiträgen ist also grundsätzlich möglich allerdings nur eingeschränkt. Ein anderes Paper wies bei der Suche des am ehesten zu einem bestimmten Autor passenden Dokumentes nur eine Accuracy von 70 Prozent auf. Dort wurde eine Strategie angewandt, bei der eine gewisse Anzahl an Ergebnissen wiedergegeben wird, die dem erstellten Profil am nächsten kommen. In diesem Fall waren es die drei nächsten und es wurde eine Genauigkeit von 90 Prozent erreicht. [5] Natürlich entsteht dabei trotz dieses Ergebnisses ein Problem, das behandelt werden muss. Und zwar ist somit noch keine eindeutige Zuordnung erfolgt, sondern nur eine Art Rangliste. Daher würde sich die Frage anschließen, wie mit dieser Rangliste verfahren werden soll.

III. Datenlage

Die zu analysierenden Quellcodes und ihre dazugehörigen Autoren aus diesem Paper wurden aus dem Programmierwettbewerb 'FIre 2020 AI-SOCO' (Authorship Identification of Source Code) bezogen, die durch den Codeforces Online Judge erfasst wurden. Dabei sollten Programmierer zu einer bestimmten Aufgabe mit einem Zeitlimit ein Programm schreiben, ohne Teilen eines schon vorgefertigten Codes zu beanspruchen. Aus allen Bewerbern wurden 1000 ausgewählt, sowie jeweils 100 Quellcodes von Ihnen. Alle Quellcodes sind "correct", was bedeutet, dass sie problemlösend und ordnungsgemäß abgegeben wurden. Des Weiteren sind sie fehlerfrei kompilierbar und in der Sprache C++ geschrieben [A]. Die Programmierer, welche die Codes ordnungsgemäß einreichten kommen aus insgesamt 78 verschiedenen Ländern. Da viele eine unterschiedliche Herkunft haben, sind die Quell-

texte in verschiedenen Sprachen geschrieben. Einige von Ihnen kommen aus dem östlichen Raum, wo ein anderes Alphabet bzw. Logogramme (Wortzeichen) statt Wörter verwendet werden, wie z.B. das kyrillische Alphabet mit 33 Buchstaben bzw. Kanjis als Zeichen, werden in den Texten verwendet. Das Dataset, mit dem gearbeitet werden sollte ist ein balancierter Datensatz, dieser wurde in Trainings- und Testdaten eingeteilt. Wobei die Verteilung bei 80 Prozent für Trainings- und 20 Prozent Testdaten waren. Jedoch wurden nicht alle Daten auf einmal freigegeben. Zu Beginn der Bearbeitung gab es insgesamt 25.000 Dokumente. Später wurden weitere 50.000 Dokumente hinzugefügt, sodass mit 75.000 Dokumente gearbeitet werden konnte. Die restlichen 25.000 Quellcodes wurden nicht freigeschalten, weshalb mit insgesamt 75.000 gearbeitet werden konnte, anstatt der 100.000 erwähnten aus dem Wettbewerb. Bei der Recherche zu diesen Quellcodes aus dem Datensatz zeigte sich, dass nicht alle Dokumente eine ähnliche Länge aufweisen. Aus den Informationen von dem Wettbewerb konnte entnommen werden, dass das längste Dokument aus 10.189 und das kürzeste aus drei Tokens besteht. Außerdem weisen alle Quellcodes und Autoren eine Kennungsnummer, auch genannt ID, auf. Die Autoren besitzen eine UID (User Identification) und zu den Quellcodes gehört eine PID (Problem Identification). Diese dient der späteren Zuordnung der Codes zu den Autoren.

IV. Methoden

Um die oben genannte Aufgabe der Zuordnung der Codes zu den Autoren zu lösen, wurde mit Hilfe der Programmiersprache R und verschiedenen SVM's an den Daten und den selbsterstellten Features gearbeitet.

A. Preprocessing

Durch das Preprocessing soll der Datensatz in eine erste Form gebracht werden, mit der später leichter gearbeitet werden kann. Als erstes wurden die CSV Daten eingelesen und die UIDs mit den PIDs verknüpft. Diese halfen die Dokumente dem richtigen Autor zuzuordnen. Auch wurde eine k-fold repeated cross validation, eine Normalisierung und Standardisierung, mittels centering und scaling und eine Zeichenumwandlung angewandt. Bei der k-fold repeated cross validation wurde der Datensatz, in k gleich große Datensatzteile eingeteilt. Ein Abschnitt davon wird als Trainingsdaten und ein anderer als Testdaten verwendet. Bei der selbstgewählten cross validation war der Anteil 80 zu 20. Dies n-mal, in diesem Fall 10-mal wiederholt. Auch werden die Datensatzteile bei jeder Wiederholung neu eingeteilt, sodass es auch andere Trainings- bzw. Testdaten gibt und diese durchgewechselt werden. Diese Wiederholung wurde drei Mal angewendet [E]. Da einige Quellcodes Zeichen wie Kanjis bzw. Hanzi und verschiedene Alphabete, wie das Kyrillische beinhalten, mussten jene Dokumente in ein anderes Format gebracht werden. Damit mit diesen gearbeitet werden konnte, mussten sie mittels UTF-8 in einen Unicode umgewandelt werden.

B. Feature Modelling

Bei dem Erstellen der Features wurde davon ausgegangen, dass dadurch viele individuelle Merkmale eines Dokumentes eruiert werden können, die auf einen speziellen Autor hindeuten. Dies würde helfen das Dokument dem richtigen Programmierer korrekt zuzuweisen. Dafür wurde als erstes eine Stichprobe von Codes betrachtet, um die Feature Bestimmung zu beschleunigen. Es wurden zu Beginn Features bestimmt, welche am ehesten auf den Autor schließen lassen. Einige davon sind die Variablennamen bzw. deren Länge, die generelle Zeichenlänge eines Quellcodes, die Häufigkeit der Verwendung von If Anweisungen, sowie for und while Schleifen und weitere Feature [siehe Tabelle], die von Autor zu Autor individuell sind. Damit wird darauf abgezielt, so viele Alleinstellungsmerkmale wie möglich in einem Dokument zu finden und diese dann einem Programmierer zuzuordnen. Es wird dabei von den groben Features wie Zeichenlänge oder Anzahl der Schleifen, zu den herausstechenden Merkmalen, wie dem Durchschnitt der meistverwendeten Zeichen in einem Code oder der Kommentarlänge, übergegangen. Dadurch entstanden letztendlich über 1.000 Feature, die helfen sollten die Quellcodes richtig zuzuweisen.

V. Ergebnisse

Bei der erstmaligen Ausführung und Verarbeitung der Daten in einer SVM, wurden vier Feature angewandt. Die LS-SVM arbeitete mit Zählen der For und While Schleifen, der If Anweisungen, sowie der Spaces, Tabs und New Lines (Tabelle I). Dabei wurden aus den insgesamt 1000 Autoren nur 20 verwendet denen ihre 25 Dokumente zugeordnet werden sollten. Dabei konnte eine Genauigkeit von 0.42 erreicht werden. Auf die LS-SVM folgte eine SVM mit radialem Kernel. In dieser wurden weitere Feature zur Untersuchung der Quellcodes eingearbeitet. Für 20 Autoren ergab sich hier eine Accuracy von 0.79, für 100 Autoren 0.63 und für 300 Autoren 0.55. Zu diesem Zeitpunkt umfasste der Datensatz 25 Dokumente je Autor. Nach der radialen SVM wurde eine SVM mit gewichtetem radialem Kernel auf dem Datensatz getestet. Mit ihr stieg die Accuracy von 0.63 auf 0.67 bei 100 und von 0.55 auf 0.57 bei 300 Autoren. Danach wurden mehrere SVM's miteinander verglichen, um die Beste herauszufinden, mit der gearbeitet werden sollte. Am Ende fiel die Entscheidung auf die Lineare SVM. Mit ihr wurden die weiteren Zuordnungen und Untersuchungen der Dokumente getätigt. Während des Testens der neuen Support Vector Machine, wurde der zweite Teil des Datensatzes zur Verfügung gestellt. Somit konnten dann statt mit 25.000 nun mit 75.000 Dokumenten gearbeitet werden. Dadurch ergab sich eine weitere Erhöhung der Accuracy. Des Weiteren wurde getestet, ob es eine Verbesserung der Accuracy gibt, bei Dokumenten mit mehr als 400 Zeichen gibt. Ein anderes Mal wurde die Genauigkeit für alle Dokumente, eingeschlossen derer unter 400 Zeichen geprüft. Dies erzielte einen bedeutenden Unterschied in der Accuracy. Für alle Dokumente, einschließlich den kurzen, gab es eine Genauigkeit von 0.69. Im Gegensatz dazu stieg die Genauigkeit ohne die kurzen Dokumente bei 100

TABLE I
FEATURE

Feature	Erklärung
Character Count	Es werden alle Character eines Dokumentes gezählt.
Blanks Ratio	Bei der Spaces, Tabs und Newline Ratio werden die unterschiedlichen Blanks jeweils zusammengefasst und durch die Anzahl der Zeichen geteilt.
Total Blanks	Ist die Gesamtanzahl an Spaces, Tabs und Newlines in den Quellcodes.
Zeichenfrequenz	Ist die Menge der Zeichen, die die Autoren in jedem Dokument verwenden.
While/If/For Ratio	Zählung der For und While Schleifen, sowie der If Anweisungen.
Variablennamenlänge	Durchschnittliche Länge aller Variablennamen.
Close/Open braces on dedicated line	Zählt ob und wie oft eine öffnende bzw. schließende Klammer auf einer separaten Zeile steht.
Einrückung Ratio	Summe aller Tabs und Spaces am Zeilenanfang geteilt durch Char Count.
Defines/Typedef	Zählen wie oft zur Definition von Variablen typedef und/oder define verwendet wird.
Wadef/Watyp Länge	Ermittelt die durchschnittliche Länge der Abkürzung einer Variable entweder mit typedef oder define.
Kommentarlänge	Durchschnittliche Menge an Kommentaren eines Dokumentes im Verhältnis zum Quelltext.
Funktionsparameterlänge	Durchschnittliche Länge aller Funktionsparameternamen.

Autoren auf 0.757 an. Zuletzt sollten alle 1000 Autoren mit allen Dokumenten in die SVM eingearbeitet werden. Damit man die Endgenauigkeit für die Zuweisung der Dokumente hat. Jedoch konnte dies nicht stattfinden, weil bei einer One vs. One Methode n über k mögliche Kombinationen berechnet werden müssen. Dies würde zu rechenintensiv werden. Die End Accuracy für alle erstellten Features bei 100 Autoren lag im Schnitt bei 0.75. Wobei die Beste von allen, die bei der gleichen Anzahl an Features und Autoren entstand, bei 0.87 lag.

TABLE II
BESTE ACCURACY

Methode	beste Accuracy	Autorenanzahl
LS-SVM	0.42	20
radiale SVM	0.79	20
radiale SVM	0.63	100
gewichtete radiale SVM	0.67	100
lineare SVM	0.72	200

VI. DISKUSSION

A. Verfahrenswahl

Zuallererst musste ein Verfahren gewählt werden, mit welchem die Features verarbeitet werden sollten. Zur Diskussion standen eine Support Vector Machine (SVM) oder ein Neuronales Netz (NN). Vorteil der SVM ist, dass sie recht schnell und mit wenigen Datensätzen arbeiten kann [B] und sie muss nicht erst, wie das neuronale Netz, lange trainiert werden. Auch gab es noch keinen SVM Ansatz, der für Quellcodes Identifikation genutzt wurde. Ein Neuronales Netz benötigt zur Bearbeitung von Daten viele Trainings- und Testdaten, damit ein gutes Ergebnis erzielt werden kann. Auch ist der Lernprozess langsamer als bei der der Support Vector Machine. [C] Schlussendlich fiel die Wahl auf die SVM, da die Umsetzung mit einem Neuronalen Netz rechenintensiver geworden wäre und diese für die Analyse des Autors eines Quelltextes scheinbar noch nicht verwendet wurde. Im Folgenden musste bestimmt werden, mit welcher Art der SVM genau gearbeitet werden sollte. Zur Auswahl standen die binäre -, die Multi-Class- und die Multi-Label Klassifikation. Die binäre Klassifikation konnte ausgeschlossen werden, weil diese nur zwei Klassen miteinander vergleichen kann. Das würde bei 1000 Autoren und somit 1000 Klassen nicht ausreichen. Im Folgenden wurde sich schlussendlich gegen die Multi Label und für die Multi Class Klassifikation entschieden, denn die Arbeitsweise dieser mittels Labels erschien im Endeffekt nicht sinnvoll. Durch dieses Verfahren hätten die Dokumente mehreren Autoren zugeordnet werden können [10]. Allerdings hätte dies die Aufgabe der eindeutigen Zuordnung der Quellcodes zu ihrem Programmierer nicht realisieren können. Des Weiteren musste entschieden werden, welche Variante der Multi Class SVM genutzt wird. Möglichkeiten waren unter anderem One-vs-One oder One-vs-All Methoden. Zunächst wurde sich auf die One-vs-One Variante konzentriert, da sie sich leichter implementieren ließ. Außerdem besitzt diese eine geringere Grauzone, da viele Hyperebenen verwendet werden. Für One-vs-All konnten keine Pakete für R gefunden werden, mit denen hätte gearbeitet werden können. Da die One-vs-One Variante zufriedenstellend funktionierte wurde entschieden, dass keine Notwendigkeit besteht für die aufwendige und zeitintensive Implementierung der One-vs-All Variante. Um die unterschiedlichen Quellcodes den Autoren zuzuordnen, werden Unterscheidungsmerkmale benötigt. Diese sollen möglichst ähnlich für dieselben Dokumente eines Autors und recht verschieden von den Quellcodes der anderen Programmierer sein. Die Umsetzung dessen erfolgt durch

sogenannte Feature. Sie sollen die individuellen Merkmale eines Quellcodes herausfiltern, um diesen dem richtigen Autor zuordnen zu können.

B. SVM-Methode

Bei der ersten Verarbeitung der Daten wurde die Least-Square Support Vector Machine (LS-SVM) verwendet, welche unerwartet mit den implementierten Features sofort funktionierte. Erst nach einiger Zeit wurde festgestellt, dass diese nicht für die Struktur der Daten geeignet ist, denn die LS-SVM ist auf die Arbeit mit spiralförmigen Daten ausgerichtet, welche in diesem Fall jedoch nicht vorlagen. Daher wurde zu einer radialen SVM gewechselt. In diese wurden alle Feature eingebunden, welche bis zu diesem Zeitpunkt erarbeitet worden waren. Mit ihr wurden die ersten sachdienlichen Accuracys für 20, 100 und 300 Autoren erstellt. Die höchsten Ergebnisse dabei lagen bei 0.63 mit 100 und 0.55 bei 300 Autoren. Bei der Recherche nach Verbesserungsmöglichkeiten der radialen SVM wurde eine gewichtete SVM mit radialem Kernel gefunden. Dabei sollten Gewichte auf die Feature verteilt werden, damit einige mehr und andere geringer in die SVM eingehen. Der Grundgedanke war, dass manche mehr Einfluss besaßen und somit entscheidender für eine korrekte Zuordnung waren und daher mehr Gewicht haben sollten. Für zufällig ausgewählte Gewichte wurden Ergebnisse für 100 und 300 Autoren erzielt bei denen die Accuracy nicht nennenswert stieg. Aus diesem Grund wurde, neben der weiteren Suche nach neuen Features, eine Analyse der verschiedenen SVM's veranlasst, um ein noch besseres Ergebnis zu erreichen. Dabei wurden mehrere SVM's getestet wobei die "svmpoly" und die "svmlinear" mit ihrer Genauigkeit herausstachen (Tabelle III). Die "svmpoly" zeigte eine geringfügig bessere Genauigkeit, jedoch benötigte sie deutlich mehr Zeit als die "svmlinear". Deswegen wurde sich letztendlich für die SVM-Linear entschieden. Mit dieser stieg die Accuracy für 200 Autoren auf 0.69. Mit allen erstellten Features stieg sie bei 100 Autoren durchschnittlich auf 0.75. Jedoch erfolgte bei einem Testlauf einen Höchstwert von 0.87. Dieses Ergebnis ist als Ausreißer zu betrachten. Solche entstehen aufgrund dessen, dass bei jedem Durchlauf ein anderer Teil des Datensatzes verwendet wird, sowohl bezogen auf die Dokumente als auch die Auswahl an Autoren. Durch so eine günstige Wahl kann eine Genauigkeit erreicht werden, die sich deutlich von den anderen abhebt und daher als Ausreißer bezeichnet wird.

TABLE III
SVM METHODEN

Methode	Accuracy SVM	Accuracy Test
SVMLinear3	0.829	0.729
LS-SVMRadial	0.827	0.709
SVMRadialWeights	0.830	0.716
SVMLinear	0.853	0.770
SVMLinear2	0.846	0.756
SVMPoly	0.854	0.763
SVMRadial	0.821	0.723
SVMRadialCost	0.824	0.723
SVMRadialSigma	0.835	0.716

C. Schwierigkeiten

Die einzelnen Klassen von dem Durchlauf für 20 Autoren wurden zunächst einmal näher betrachtet. Dabei wurde festgestellt, dass einige nur eine Zuordnungsgenauigkeit von 0.59 besitzen, was recht gering ist. Im Gegenzug gibt es auch Klassen, die eine Genauigkeit von 1.00 besitzen was aussagt, dass alle Dokumente korrekt zugeordnet wurden. Die Frage ist nun, wodurch diese unterschiedlich genaue Zuordnung einzelner Dokumente zu einem Autor bedingt sein kann. Eine logische Ursache könnte eine kurze Dokumentenlänge sein. Deshalb wurde in einem Testlauf nur auf Dokumente eingegangen, die eine Zeichenanzahl über 400 besitzen. Es ergab sich, dass die Accuracy bei 200 Autoren von 0.69 um 0.3 anstieg. Daran lässt sich erkennen, dass die kurzen Dokumente eine Fehlerquelle sind. Sie lassen sich schlechter untersuchen, da sie weniger Erkennungsmerkmale besitzen und somit können nur wenige Feature ausreichend stark herausgefiltert werden. Dafür musste eine Lösung gefunden werden welche letztendlich daraus bestand, dass die kurzen Dokumente aus dem Trainingsdatensatz entfernt wurden, aber im Testdatensatz weiterhin vorhanden sind. Der Grund dafür ist, dass mit Ihnen schlechter trainiert werden kann, sie aber dennoch einem Autor zugewiesen werden müssen. Als der Durchlauf für 300 Autoren intensiver betrachtet wurde, konnte allerdings festgestellt werden, dass bei dem Autor mit der PID 73 die Accuracy recht niedrig war, obwohl die Länge der Dokumente bei über 400 Zeichen lag. Daraus konnte festgestellt werden, dass nicht viele Alleinstellungsmerkmale erfasst wurden. Dadurch konnte bewiesen werden, dass es notwendig ist neue Feature zu implementieren. Dies ist wichtig um die richtige Zuordnung der Dokumente zu ihren Autoren zu steigern und damit die Accuracy weiter zu verbessern. Des Weiteren gab es Probleme bei der Umwandlung der verschiedenen Zeichen. Zum einen wurden z.B. Kanjis (japanische Zeichen) nicht korrekt umgewandelt. Zum anderen gab es Probleme bei dem Umgang mit den Zeichen für Alpha und Pi. Das Problem lag darin, dass R diese Zeichen ebenfalls nicht richtig umwandelte. Das Alpha wurde als A und das Pi als P betrachtet. Somit wurden dann A und P nicht mehr gewertet, da es schon einmal benannt wurde. Dieses Problem wurde umgegangen, indem die Alphas und die Pi's herausgefiltert wurden und somit nicht doppelt gewertet wurden.

D. Anhaltspunkte zur Verbesserung

Um das Ergebnis der SVM in Zukunft zu verbessern, könnten folgende Dinge ausprobiert bzw. implementiert werden. Als erstes könnte erprobt werden, wie das Ergebnis beeinflusst wird, wenn z.B. die Funktionsparameter als Feature herausgenommen werden. Falls kein nennenswerter Unterschied zu verzeichnen wäre, könnte dieses Feature ausgeklammert und sich auf andere konzentriert werden. Dadurch ließe sich an Rechenleistung sparen. Darauffolgend könnten die Variablennamen als eigenes Feature für alle Variablen im Dokument in die SVM eingehen. Zurzeit wird nur der Durchschnitt aller Variablennamen gebildet, weil die Rechenkapazität ausgelastet ist. Zudem kann ein Abstract Syntax Tree (AST)

als ein neues Feature erstellt werden, da die Extraktion mit Hilfe von Suchmustern durchgeführt wurde. Dabei wird der Quelltext als Baum dargestellt. Einige Compiler können einen AST erstellen wodurch es sich anbietet diesen dann auch bei den Analysen des Quelltexts zu benutzen. Interpretierte Programmiersprachen setzen bei der Verarbeitung dagegen, aufgrund der besseren Performance, andere Darstellungsformen ein. Diese Arbeit sollte sich auch leicht für andere Programmiersprachen implementieren lassen weshalb sich entschieden wurde Suchmuster zu benutzen. Eine abgewandelte Version dieser Arbeit, die sich auf eine Programmiersprache spezialisieren soll, kann dagegen einen durch den Compiler erstellten AST benutzen, um auch künftige Änderungen in der Sprache leicht implementieren zu können. Wenn dies in der One vs One Methode ausprobiert wurde, könnte versucht werden eine One vs All Variante zu erstellen, um die Rechenkapazität zu verringern. Mit der dann neuen geschaffenen Rechenkapazität könnten mehr Feature implementiert und eine höhere Accuracy erwartet werden.

VII. Zusammenfassung

Authorship Analysis, Source Code Identification ist in der heutigen Zeit ein wichtiger Aspekt zur Analyse von Quellcodes. Die Umsetzung dessen, mit Hilfe der SVM und den selbsterstellten Features zeigte, dass es ein brauchbarer Lösungsansatz für die zu lösende Problemstellung ist. Es wurde eine durchschnittliche Best-Accuracy bei 100 Autoren von 0.75 erzielt. Dies überschreitet zwar die Anfangs-Accuracy von 0,42 und die Baseline von 0,62 für KNN, jedoch werden nicht alle Dokumente und Autoren verwendet, weil die Rechenkapazität nicht ausreichte. Die dabei am Ende verwendete Methode der SVM war die Lineare Support Vector Machine mit den vorher aufgeführten Features und 100 Autoren mit je 40 Dokumenten. Einmal wurden alle Dokumente betrachtet und ein anderes Mal nur die, welche eine Dokumentenlänge von über 400 Zeichen besitzen. Es hat gezeigt das kurze Dokumente schwieriger zuzuordnen waren, als längere. Auch spielt die Anzahl der Feature eine Rolle. Je wenige Feature, desto weniger können die Quellcodes den Programmierern zugeordnet werden. Verglichen mit Papern, in denen als Lösungsansatz ein Neuronales Netz benutzt wird, lagen wir unter deren Accuracy von über 0.9.[2, 14]. Es zeigt sich also, dass die SVM zur Analyse des Autors eines Quellcodes konkurrenzfähig ist, jedoch noch mehr Features benötigt werden. Sie erfordert zwar mehr Rechenkapazität als erwartet aber trotzdem weniger als beispielsweise Neuronale Netze. Abschließend lässt sich sagen, dass das Projekt ein befriedigendes Ergebnis erzielt hat. Allerdings besteht noch Verbesserungspotential indem zum Beispiel die erwähnten Anhaltspunkte zur Verbesserung umgesetzt werden könnten. Aufgrund der geringeren Rechenintensität eignet sich die SVM durchaus als Lösungsansatz für die Source Code Identification stößt aber in gewissen Bereichen dennoch auf ihre Grenzen.

References

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first . . .”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.