Leibniz Universität Hannover

Fakultät für Elektrotechnik und Informatik
Institut für Verteilte Systeme
Fachgebiet Wissensbasierte Systeme (KBS)

Masterarbeit

Entdeckung von Spammern in kollaborativen Verschlagwortungssystemen

Spammer Detection in Collaborative Tagging Systems

Christian Kater

Erstprüfer Prof. Dr. rer. nat. Robert Jäschke
Fachgebiet Wissensbasierte Systeme (KBS)
Leibniz Universität Hannover

Zweitprüfer Prof. Dr. techn. Wolfgang Nejdl
Fachgebiet Wissensbasierte Systeme (KBS)
Leibniz Universität Hannover

Betreuer Prof. Dr. rer. nat. Robert Jäschke

17. Mai 2016

Erklärung

Hiermit versichere ich, dass ich die vorliegende Abschlussarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, 17. Mai 2016

Christian Kater

Zusammenfassung

Spam ist ein weit verbreitetes Problem für viele Online-Services. In dieser Arbeit wird das kollaborative Verschlagwortungssystem Bibsonomy betrachtet, welches über 150 mal mehr Registrierungen von Spammern entgegennahm, als von normalen Benutzern über die letzten 10 Jahre.

Ein üblicher Ansatz zur Bekämpfung von Spam ist die Verwendung von Machine-Leanring-Verfahren. Mit diesen werden die Benutzer als Spammer oder normale Benutzer klassifiziert. Dabei werden aus den Informationen, die der Benutzer in seinem Profil hinterlegt oder durch von Beiträge offen gelegt hat, Features erzeugt, mit denen der Klassifikator seine Entscheidung treffen kann. Oftmals bedeutet das jedoch auch, dass Spammer bereits ihre Inhalte im System verbreiten.

In dieser Arbeit wird ein Lösungsansatz zur Klassifikation von Benutzern, als Spammer oder normale Benutzer, zur Registrierung bzw. zur Aktivierung vorgestellt. Dazu wurden 270 Feature aus den Informationen, die der Benutzer zur Registrierung angegeben hat, dessen IP-Adresse, das Datum der Registrierung, sowie Informationen wie die Profilinformationen während der Registrierung eingegeben wurden, extrahiert. Mit diesen Features wurden state-of-the-art Klassifikatoren benutzt, um die Benutzer als Spammer oder normale Benutzer zu klassifizieren. Der beste Klassifikator erreichte dabei einen AUC von 0,91.

Abstract

Spam is a widespread problem for many online services. The use case in this thesis is the social bookmarking system BibSonomy, which received over 150 times more registrations from spam users than from normal users over the last ten years.

A common approach to fight spam is to use machine learning to classify the users into good or malicious users. Based on information the users provide to the service in form of profile information or posts, features are created from which a classifier can make its decision. However, this often means that the accounts of the spam users are already active and can post their spam.

In this thesis an approach for deciding at registration time whether a user is malicious or not is proposed. In order to achieve this goal, we extracted 270 features from the information the users provide during the registration process, their IP address, registration time, and informations based in how they entered the informations. With these features state-of-the-art classifiers was used to identify users as spammers or regular users. With the best classifier an AUC of 0.91 could be reached.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **3**
2.1 Spammer . . . 3
2.2 Klassifikation . . . 3
2.2.1 Naive Bayes . . . 4
2.2.2 Support Vector Machine . . . 5
2.2.3 J48 . . . 5
2.2.4 Logistic Regression . . . 6
2.3 Messgrößen . . . 6
2.3.1 Receiver Operating Characteristic . . . 7
2.4 Statistik Tests . . . 9

3 Verwandte Arbeiten **11**

4 Datensatz **13**
4.1 Bibsonomy Benutzer Tabelle . . . 13
4.2 Interaktionen bei der Registerierung . . . 14
4.2.1 Event-Logger . . . 16
4.3 Apache-Log . . . 17

5 Features **21**
5.1 Sprach-basierte Features . . . 22
5.2 Umwelt-basierte Features . . . 29
5.3 Populations-basierte Features . . . 32
5.4 Tastatur-basierte Features . . . 34
5.5 Auf Log-Daten basierende Features . . . 38
5.6 Interaktions-basierende Features . . . 41

6 Methodik **45**
6.1 Klassifikatoren . . . 45
6.2 Feature Selection . . . 45
6.3 Class Imbalance . . . 46
6.4 Evaluation . . . 46

7 Ergebnisse **51**

8 Anbindung an Bibsonomy **59**
8.1 Benchmark 61

9 Fazit **63**

A Anhang **65**
A.1 Sprach-basierte Features 65
A.2 Umwelt-basierte Features 72
A.3 Populations-basierte Features 73
A.4 Tastatur-basierte Features 76
A.4.1 Auf Log-Daten basierende Features 94
A.4.2 Interaktions-basierende Features 96

Literatur **109**

1 Einleitung

Bibsonomy ist ein kollaborative Verschlagwortungssystem, das Benutzern ermöglicht Lesezeichen und Publikationen online zu verwalten und mit anderen Benutzern zu teilen, sowie diese mit Tags zu versehen. Unglücklicherweise sind solche Systeme sehr attraktiv für Webspam. Benutzer teilen Lesezeichen oder Veröffentlichungen, die auf kommerzielle oder dubiose Seiten verweisen, um dadurch im Ranking von Suchmaschinen aufzusteigen und die Besucherzahlen ihrer Webseiten zu verbessern. Diese Maßnahmen werden mit dem Ziel durchgeführt, höhere Umsätze, durch Werbung oder Produktverkäufe, zu erzielen. Ein Einstellen von Lesezeichen oder Publikationen mit den eben genannten Zielen ist, laut Allgemeinen Geschäftsbedingungen von Bibsonomy, untersagt. Dennoch hindert das, die wenigsten daran Spam in Bibsonomy zu verbreiten. Ein hohes Maß an Spam in Bibsonomy kann dazu führen, dass die Benutzerfeundlichkeit in Bibosonomy sinkt. Bibsonomy lebt letztlich von seinen Inhalten. Je mehr irrelevant Inhalte vorhanden sind, desto schwerer ist es für die interessanten und relevante Inhalte herauszustechen. Daher sind geeignete Maßnahmen zur Verhinderung von Webspam zwingen notwendig.

Seit 2008 wurde ein Spam-Detektions-Framework entwickelt, welches Machine-Learning-Verfahren benutzt, um einen Benutzer anhand seiner Beiträge zu Bibsonomy zu klassifizieren [21]. Beiträge von Benutzern, die als Spammer markiert wurden, sind vor anderen Benutzern versteckt. Der dabei benutzte Datensatz wurde / wird von dem Team von Bibsonomy verwaltet und gepflegt. Die Markierungen von Benutzrn als Spammern wird dabei von den Administratoren von Bibsonomy vorgenomen. Der Nachteil des bereits implementierten Frameworks ist es, dass Spammer erst als solche erkannt werden, wenn diese bereits Spam zu Bibsonomy hinzugefügt haben. Daher können Maßnahmen zur Reduzierung von Spam Beiträgen, wie das ausfüllen eines Captchas beim hinzufügen eines neuen Beitrags, nicht direkt greifen.

In dieser Arbeit sollen Benutzer bereits zur Registrierung, bzw. zur Aktivierung ihres Accounts, klassifiziert werden. Dadurch sollen Spammer bereits, als solche markiert werden, bevor diese überhaupt Spam zu Bibsonomy hinzufügen können. Dabei werden Angaben, die der Benutzer bei der Registrierung macht, aber auch die Art und Weise, wie diese Angaben eingeben werden untersucht und als Features für die Klassifikation bereitgestellt. Ebenso sollen die gefundenen Features und

Ergebnissen, ebenso wie eine Anbindung an Bibsonomy, implementiert werden, so dass Bibsonomy möglichst Zeitnah von diesen profitieren kann.

Der Vorteil einer Klassifizierung zur Aktivierung liegt vor allem darin, dass mögliche Gegenmaßnahmen zu einem frühstmöglichen Zeitpunkt eingeleitet werden können. Gewisse Maßnahmen, wie z.B. ein Tutorial zur Bedienung, das für alle Spammer verpflichtend ist und somit ein gewisses Hindernis, im Sinne von aufzuwendender Zeit, darstellt, sind auch nur direkt nach der Aktivierung sinnvoll. Zudem können bereits vorhandene Maßnahmen früher greifen und somit helfen, die Anzahl an neuen Spam-Beiträgen gering zu halten.

Die vorliegende Arbeit unterteilt sich in die Einleitung und acht weiteren Kapiteln. Im zweiten Kapitel werden zunächst grundlegende Begriffe und Methoden zur Klassifikation und dessen Auswertung dargestellt. Im nachfolgenden Kapitel wird Bezug zu ähnlich gelagerten Arbeiten genommen und Gemeinsamkeiten und Anknüpfungspunkte diskutiert. Im vierten Kapitel werden die benutzten Datensätze und deren Erzeugung beschrieben und erläutert. Im fünften Kapitel werden die für die Klassifikation benutzen Features vorgestellt. Es werden ihre Funktionsweise erläutert, begründet und mittels Diagramme visualisiert, warum es sinnvoll ist diese einzusetzen. Im sechsten Kapitel werden Methoden zur Evaluation der gefunden Features und weiteren Maßnahmen zur Erhöhung der Klassifikations-Güte beschrieben. Im anschließenden Kapitel werden dessen Ergebnisse ausgewertet und diskutiert. Im achten Kapitel wird die technische Anbindung an Bibsonomy beschrieben. Zuletzt werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick über zukünftige Arbeiten gegeben.

Ein Teil dieser Masterarbeit wurde, zusammen mit Prof. Dr. rer. nat. Robert Jäschke, als wissenschaftliche Veröffentlichung, unter dem Titel "You Shall Not Pass: Detecting Malicious Users at Registration Time", auf dem Workshop Online Safety, Trust and Fraud Prevention der Webscience Konferenz 2016 in Hannover eingereicht und akzeptieret. Die Ergebnisse der Veröffentlichung, und somit auch dieser Arbeit, werden am 22. Mai 2016 auf dem besagten Workshop vorgestellt und mit der anwesenden wissenschaftlichen Community diskutiert.

2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Konzepte, die zum Verständnis dieser Arbeit notwendig sind, vorgestellt. Ausgehend von der Definition eines Spammers, wird das Grundprinzip der Klassifikation, sowie die grundlegenden Funktionsweisen der eingesetzten Klassifikatoren, erläutert. Anschließend werden grundlegende Messgrößen und Konzepte zur Bewertung von Klassifikatoren vorgestellt. Zuletzt wird der Kolmogoroff-Smirnoff-Test erläutert. Dieser wird in einem der nachfolgenden Kapitel dazu benutzt, um zu entscheiden, ob sich die Werte-Verteilungen zwischen Spammern und Nicht-Spammern signifikant unterscheiden.

2.1 Spammer

Das Wort Spam wird häufig mit unerwünschten, meist kommerziellen, Nachrichten in großen Mengen assoziert. Die Häufigste Verbreitungsform sind hierbei Spam über E-Mails [8]. Die Art von Spam, der Bibsonmy gegenüber steht, ist Webspam. Webspam sind die Maßnahmen, die die Relevanz einer Webseite ungerechtfertigterweise höher erscheinen lassen, als diese tatsächlich ist [13]. Als Spammer werden dabei solche Benutzer angesehen, die Spam als Links oder Beiträge verbreiten. Ob ein Beitrag oder Link Spam, und damit dessen Verursacher ein Spammer ist, ist durchaus subjektiv. Ein Beitrag, der für eine Person Spam ist, kann für eine andere Person noch akzeptabel sein. Letztlich gilt für diese Arbeit ein Benutzer als Spammer, wenn die Administratoren von Bibsonomy diesen als Spam angesehen und entsprechend markiert haben.

2.2 Klassifikation

Als Klassifikation wird das Problem bezeichnet für eine neue Beobachtung, aus einer Menge von Klassen, eine Zuordnung zu finden. Im Kontext dieser Arbeit muss für einen Benutzer entschieden werden, ob dieser der Klasse Spammer oder der Klasse Nicht-Spammer zuzuordnen ist. Um diese Entscheidung zu treffen wird eine Beobachtung, bzw. hier ein Benutzer, durch eine Menge von Eigenschaften beschrieben. Diese Eigenschaften werden Features genannt. Beschreiben mehrere Features das gleiche Phänomen, jedoch angewendet auf verschiedene Bereiche der Beobachtung, so werden diese zu einem Meta-Features zusammengefasst. Würden

beispielsweise je ein Feature beschreiben, dass der Benutzername, die E-Mail-Adress und der Vorname eines Benutzers mit einem A beginnt, so könnte alle drei Features zu einem Meta-Feature zusammengefasst werden. Alle diese Features geben an, ob ein Wort mit einem A beginnt. Angewendet werden diese jedoch auf verschiedene Eigenschaften des Benutzers.

Algorithmen zur Lösung dieses Problems werden Klassifikatoren genannt. Ein Klassifikator besteht aus zwei Phasen. In der ersten Phase lernt der Klassifikator anhand einer Menge von Beobachtungen und dessen Zugehörigkeit zur jeweiligen Klasse. In der zweiten Phasen kann der Klassifikator mit diesem Wissen für neue Beobachtungen entscheiden zu welcher Klasse diese zuzuordnen ist.

Als Grundlage für die Klassifikation wurde Weka[1] gewählt. Weka ist eine von der Machine Learning Group der University of Waikato gepflegte Software-Bibliothek von Machine-Learning-Algorithmen.

Im nachfolgenden werden in der Spam-Erkennung übliche und in dieser Arbeit verwendete Klassifikatoren vorgestellt und deren grundlegende Funktionsweise erklärt.

2.2.1 Naive Bayes

Der Naive Bayes Klassifikator benutzt die bedingten Wahrscheinlichkeiten der Klassen und Feature und den Satz von Bayes, um einer Beobachtung eine Klasse zuzuordnen. Sei K_{MAP} die meist wahrscheinlichste Klasse (Maximum a Posteriori) aller Klassen K und B die zu klassifizierende Beobachtung. K_{MAP} ist wie folgt definiert:

$$K_{MAP} = \arg\max_{k \in K} P(k|B) = \arg\max_{k \in K} \frac{P(B|k) \cdot P(k)}{P(B)} = \arg\max_{k \in K} P(B|k) \cdot P(k)$$

Eine Beobachtung besteht aus mehreren Features. Es wird angenommen, dass alle Feature statistisch unabhängig voneinander sind. Daher kann $P(B|k)$ als Multiplikative Summe der bedingten Wahrscheinlichkeiten der Features berechnet werden. Sei F die Menge aller Features in B. Dann ergibt sich K_{MAP} wie folgt:

$$K_{MAP} = \arg\max_{k \in K} P(k) \cdot \prod_{f \in F} P(f|k)$$

$P(f|k)$ und $P(k)$ können dabei sehr einfach durch Zählen der Werte in den Trainings-Daten ermittelt werden.

[1] http://www.cs.waikato.ac.nz/ml/weka/index.html

2.2.2 Support Vector Machine

Die Support Vector Maschine (SVM) betrachtet die zu klassifizierenden Beobachtungen als Vektoren. Dabei werden die Vektoren zweier Klassen unterschieden Die Grundlegende Idee der Support Vector Machine ist es in diesem Vektorraum eine Hyperebene zu finden, die den Abstand zwischen den Vektoren beider Klassen maximiert. Dabei betrachtet die SVM jedoch nur solche Vektoren, die den Rand der Hyperebene berühren (siehe Abbildung 2.1). Diese Vektoren werden Stützvektoren (englisch: support vector) genannt. Um auch Nicht-Linear-Trennbare Daten separieren zu können werden die Vektoren in einer höher Dimensionalen Raum überführt, sodass die Vektormenge linear trennbar wird. Funktionen, die die Hyperebene beschreiben, werden Kernelfunktionen genannt.

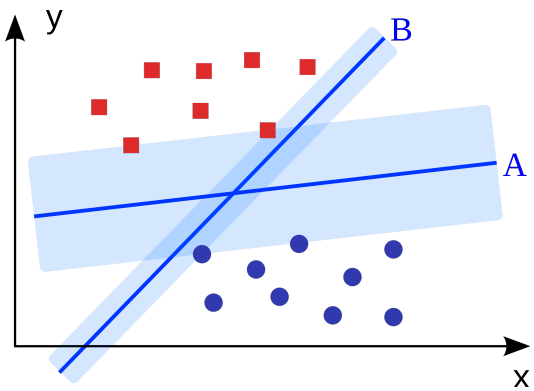


Abb. 2.1.: Beispiel SVM mit 2-dim. Daten und linearer Kernelfunktion. Quelle: [@28]

2.2.3 J48

J48 ist eine freie Java Implementierung des C4.5 Algorithmus. Der C4.5 Algorithmus ist ein auf einem Entscheidungsbaum basierender Klassifikator. Mit den Trainingsdaten wird ein solcher Entscheidungsbaum aufgebaut, dessen Blättern die entsprechende Klassen sind. Um diesen Entscheidungsbaum aufzubauen werden die Trainingsdaten Anhand eines Features immer wieder aufgeteilt. Diese Aufteilung entspricht einem Knoten im Entscheidungsbaum. Das Feature mit dem höchsten Informationsgewinn, wird als nächstes zum Splitten der Daten benutzt.

Um eine Beobachtung zu klassifizieren muss entsprechend der Werte der Feature ein Pfad im Baum bis zu einem Blatt gelaufen werden. Entsprechend der Klasse des Blattes wird auch die Beobachtung klassifiziert.

2.2.4 Logistic Regression

Logistic Regression ist in Klassifikator, der Beobachtungen mittels Regressionsmodellen klassifiziert. Dabei werden die Parameter dieses Modells mittels der Trainingsdaten bestimmt. Neuen Beobachtungen können so eine Wahrscheinlichkeit für jede Klasse zugewiesen werden, womit die wahrscheinlichste Klasse bestimmt werden kann.

2.3 Messgrößen

Dieser Abschnitt definiert die üblichen Messgrößen zur Bestimmung der Güte eines Klassifikators. Diese basieren auf den richtig bzw. falsch vorhergesagten Klassen. Klassifiziert ein Klassifikator einen Nicht-Spammer als Nicht-Spammer, so spricht man von einer Richtig-Negativen Vorhersage. Wird ein Spammer als Nicht-Spammer klassifiziert, so spricht man von einer Falsch-Negativen Vorhersage. Entsprechendes gilt für den Fall, dass Spammer als Spammer (Richtig-Positiv) und Nicht-Spammer als Spammer (Falsch-Positiv) klassifiziert werden. Tabelle 2.1 zeigt eine Wahrheitsmatrix eines Klassifikators. Diese enthält nach der Evaluation eines Klassifikators die Anzahl der Richtig-Negativ, Falsch-Negativ, Falsch-Positiv und Richtig-Positiv getätigten Vorhersagen.

Tab. 2.1.: Wahrheitsmatrix eines Klassifikators

		Tatsächlich	
		Nicht-Spammer	Spammer
Vorhersage	Nicht-Spammer	Richtig-Negativ (RN)	Falsch-Negativ (FN)
	Spammer	Falsch-Positiv (FP)	Richtig-Positiv (RP)

Basierend auf den Einträgen der Wahrheitsmatrix werden die folgenden Kriterien zur Bestimmung der Güte eines Klassifikators, wie folgt definiert.

Definition 2.1 *Richtig-Positiv-Rate*
Die Richtig-Positiv-Rate (englisch: true positive rate) eines Klassifikators ist das Verhältnis von korrekt klassifizierten Spammer zu Spammern insgesamt. Sie ist definiert durch [11]:
Richtig-Positiv-Rate $= \frac{RP}{RP+FN}$

Definition 2.2 *Falsch-Positiv-Rate*
Die Falsch-Positiv-Rate (englisch: false positive rate) eines Klassifikators ist das Verhältnis von falsch klassifizierten Nicht-Spammern zu Nicht-Spammern insgesamt. Sie ist

definiert durch [11]:
Falsch-Positiv-Rate $= \frac{FP}{RN+FP}$

Definition 2.3 *Precision*
Die Precision im Sinne des Information Retrieval ist das Verhältnis von relevanten zu erhaltenen Dokumenten. Angewendet auf die Klassifikation ist die Relevanz das Verhältnis korrekt klassifizierter Instanzen einer Klasse zu allen als diese Klasse klassifizierten Instanzen. Sie ist definiert durch [11]:
Precision (Spammer) $= \frac{RP}{RP+FP}$
Precision (Nicht-Spammer) $= \frac{RN}{RN+FN}$

Definition 2.4 *Recall*
Der Recall im Sinne des Information Retrieval ist das Verhältnis von relevanten zu relevanten Dokumenten insgesamt. Angewendet auf die Klassifikation ist der Recall die Richtig-Positiv-Rate für Spammer, bzw. die Richtig-Negativ-Rate für Nicht-Spammer. Sie ist definiert durch [11]:
Recall (Spammer) = Richtig-Positiv-Rate $= \frac{RP}{RP+FN}$
Recall (Nicht-Spammer) = Richtig-Negativ-Rate $= \frac{RN}{RN+FP}$

Definition 2.5 *F1-Maß*
F1-Maß setzt Precision und Recall in Beziehung. Es ist definiert durch [22]:
F1-Maß $= 2\frac{Precision \cdot Recall}{Precision+Recall}$

2.3.1 Receiver Operating Characteristic

Nicht immer sind Falsch-Positiv und Falsch-Negativ gleich schwerwiegend. Im Falle dieser Arbeit ist ein Nicht-Spammer, der als Spammer angesehen wird, schädlicher als ein Spammer, der als Nicht-Spammer angesehen wird. Denn durch die falsche Klassifizierung des Nicht-Spammers geht unter umständen ein potenzieller Nutzer von Bibsonomy verloren. Bei einem falsch klassifizierten Spammer hingegen kann zwar in Bibsonomy Spam verbreiten, jedoch kann er immer noch von dem bereits implementierten Framework erfasst werden, welches Spammer unter anderem durch ihre Beiträge klassifiziert. Eine Möglichkeit Einfluss auf die Falsch-Positiv-Rate zu nehmen sind Klassifikatoren, die neben der Angabe der Klasse, auch die Wahrscheinlichkeit zur Zugehörigkeit zur Positiven Klasse angeben können. Mit dieser Wahrscheinlichkeit muss nicht mehr die wahrscheinlichste Klasse gewählt werden, sondern es kann eine Grenzwahrscheinlichkeit, ab der ein Benutzer als Spammer gilt, angegeben werden. Durch die Angabe einer höheren Grenzwahrschienlichkeit kann die Falsch-Positiv-Rate herab gesetzt werden. Dadurch sinkt jedoch auch die

Richtig-Positiv-Rate. Übertragen auf das Problem der Arbeit ist also eine Abwägung zwischen wie viele Spammer und wie viele Nicht-Spammer werden von Bibsonomy ausgeschlossen. Um diesen Trade-Off zu visualisieren gibt es Receiver Operating Characteristic Graphen.

Algorithmus 1 ROC-Kurve erzeugen nach [11]

```
1:  procedure CREATEROC(L, f(i), N, P)
2:      L_sorted ← L sorted decreasing by f scores
3:      FP ← TP ← 0
4:      R = ⟨⟩
5:      f_prev ← −∞
6:      i ← 1
7:      while i ≤| L_sorted | do
8:          if f(i) ≠ f_prev then
9:              push (FP/N, TP/P) onto R
10:             f_prev ← f(i)
11:         end if
12:         if L_sorted[i] is a positive example then
13:             TP ← TP + 1
14:         else
15:             FP ← FP + 1
16:         end if
17:     end while
18:     push (FP/N, TP/P) onto R
19: end procedure
```

Receiver Operating Characteristic (ROC) Graphen sind zweidimesnionale Graphen, in denen auf der X-Achse die Falsch-Positiv-Rate und auf der Y-Achse die Richtig-Positiv-Rate dargestellt ist. Eine ROC-Kurve gibt dabei in diesem Trade-Off zu jeder Falsch-Positiv-Rate die entsprechende Richtig-Positiv-Rate an. Algorithmus 1 gibt an, wie diese ROC-Kurven zu erzeugen sind. Die Idee des Algorithmus ist die folgende: Jeder Instanz, im zu evaluierenden Datensatz, ist eine Wahrscheinlichkeit zur Zugehörigkeit zur positiven Klasse vom Klassifikator zugeordnet worden. Zusätzlich besitzt der Trainingsdatensatz für jede Instanz das entsprechende Klassenlabel. Werden diese Label nun absteigend nach ihrer Wahrscheinlichkeit sortiert, so kann an jeder Position der Sequenz eine Grenzwahrscheinlichkeit festgelegt werden. Schließlich ist jedem Klassen-Label eine Wahrscheinlichkeit zugeordnet worden. Da alle vorherigen Klassen-Label als Positiv klassifiziert werden, kann die Richtig-Positiv-Rate bzw. die Falsch-Positiv-Rate durch zählen der richtigen bzw. falschen Klassifikation vor dieser Grenzwahrscheinlichkeit und normieren mit der Anzahl an Positiven bzw. Negativen Klassen-Label insgesamt, angegeben werden. Wird die Sequenz Position für Position durchlaufen, so erhält man alle Falsch-Positiv-Rate zu Richtig-Positiv-Rate Punkte der ROC-Kurve. [11]

Diese Idee benötigt jedoch noch eine Modifikation. Wenn mehrere Klassen-Label die gleiche Wahrscheinlichkeit haben, wie werden diese angeordnet? Würden zuerst alle positiven Klassen-Label einsortiert werden, wäre die ROC-Kurve zu optimistisch, da es den bestmöglichen Fall darstellen würde. Würde erst alle negativen Klassen-Label einsortiert werden, wäre die ROC-Kurve zu pessimistisch. Da man die, der Wirklichkeit am ehesten kommende, Anordnung nicht angeben kann, wird dieser Abschnitt gemittelt. Dieses wird dadurch erreicht, dass man für diese Klassen-Label keine Punkte in die ROC-Kurve aufnimmt. Dadurch wird der zuletzt unterscheidbare Wert, mit dem nächsten Unterscheidbaren Wert verbunden. [11]

Die Güte der ROC-Kurve kann über die Fläche unter der ROC-Kurve angegeben werden. Dieses Maß wird AUC (Englisch für Area under Curve) genannt. Der perfekte Klassifikator hätte einen AUC von 1. Das würde bedeuten, dass alle positiven Klassen-Label eine höhere Wahrscheinlichkeit, zur Klasse Positiv zu gehören, haben, als alle negativen Klassen-Label. Das würde dazu führen, dass man genau eine Grenzwahrscheinlichkeit finden kann, sodass die Richtig-Positiv-Rate gleich Eins ist. Ein Klassifikator, der die Wahrscheinlichkeiten zur Zugehörigkeit zur positiven Klasse, zufällig mit Eins oder Null belegt, würde bei einem unendlichen großen Datensatz einen AUC von 0,5 haben. Das heißt, dass alle Klassifikatoren mit einem AUC von größer als 0,5 besser klassifizieren als der Zufall. Ein schlechterer AUC als 0,5 kann immer vermieden werden durch umdrehen der Entscheidungen. Alle zugeordneten Wahrscheinlichkeiten p werden zu $1 - p$. Dadurch wird auch der AUC zu $1 - AUC_{alt}$.

2.4 Statistik Tests

Um zu überprüfen, ob die Beobacheteten Daten zweier Stichproben der gleichen Verteilung folgen, kann der Kolmogoroff-Smirnoff-Test verwendet werden.

Definition 2.6 *Empirische Verteilungsfunktion [14]:*
Seien $x_1 ... x_n$ Beobachtungswerte einer Stichprobe. Dann ist die empirische Verteilungsfunktion wie folgt definiert:
$F_n(x) = \frac{1}{n} \cdot \sum_{i=1}^{n} 1_{x_i \leq x}$,
mit $1_{x_i \leq x} = 1$, falls x_i kleiner gleich x, sonst Null.

Definition 2.7 *Komplementäre Empirische Verteilungsfunktion [14]:*
Seien $x_1 ... x_n$ Beobachtungswerte einer Stichprobe. Dann ist die komplementäre empirische Verteilungsfunktion wie folgt definiert:
$K_n(x) = \frac{1}{n} \cdot \sum_{i=1}^{n} 1_{x_i > x} = 1 - F_n(x)$,
mit $1_{x_i > x} = 1$, falls x_i größer als x ist, sonst Null.

Definition 2.8 *Kolmogorow-Smirnow-Test [14]:*
Seien X und Y Stichproben mit n bzw. m Werten und $F_X(X)$ und $F_Y(X)$ die entsprechenden (komplementären) empirische Verteilungsfunktion.
So sei $d((z) =\mid F_1(X) - F_2(X) \mid$ und $d_{max} = \sup_z d(z)$
Es liegt ein signifikanter Unterschied zwischen den beiden Verteilungsfunktionen vor, wenn gilt:
$\sqrt[2]{\frac{nm}{n+m}} d_{max} > \sqrt[2]{\frac{\ln(\alpha)}{2}}$,
unter dem Signifikanzniveau α

3 Verwandte Arbeiten

Die Entdeckung von Spam ist im Fokus vieler wissenchaftlicher Arbeiten, vor allem im Bereich von Spam-E-Mails [5]. Seit 2008 wurde von der Gruppe um Bibsonomy ein Framework zur Erkennung von Spammern in Bibsonomy entwicklet [21, 2, 7]. In Krause et. al. [21] wurden die Features für dieses Framework in folgende Kategegorien unterteilt: Profil, Standort-, Aktivität-basierte, sowie semantische Features. Profil-basierte Features benutzen Informationen, die der Benutzer bei der Registrierung von Bibsononmy angegeben hat. Es wurden Meta-Features benutzt, die die Anzahl der Zeichen, sowie die Anzahl der Zahlen zählen. Diese wurden auf den Benutzernamen, E-Mail und Echtnamen angewendet. Zudem wurde betrachtet, ob der Echtname des Benutzers aus zwei Namen (Vor- und Nachname), sowie aus drei Namen (Vor-, Mittel- und Nachname) besteht. Standort-basierte Feature beziehen sich auf den Ort bzw. das Land aus dem der Benutzer aus sich bei Bibsonomy registriert hat. Zum einen wurde das Aufkommen von anderen Benutzern mit der selben Domain, sowie das Aufkommen von anderen Benutzern mit der selben Top-Level-Domain gezählt. Zum anderen wurde das Aufkommen von Benutzern, die als Spammer markiert sind, mit der selben IP-Adresse gezählt. Aktivitäts-basierte Features betrachten verschiedene Interaktionen mit Bibsonomy, wie z.B. die Zeitdifferenz zwischen Registrierung und ersten Post. Semantische Features benutzen die Tags in Bibsonomy und wie Benutzer über diese verknüpft sind. Ein Beispiel für ein Semantisches Feature ist das Aufkommen der Verbindung des Benutzers zu Spam-Benutzern über gemeinsame Tags. Insgesamt erreichen Krause et. al. [21] als bestes Ergebnis, mit SVM als Klassifikator, einen AUC von 0,936. Die notwendigen Information der Profil- und Standort-basierten Features sind bereits nach der Registrierung vorhanden. Daher wurden diese Features in diese Arbeit übernommen. Die Features der anderen beiden Kategorien benutzen Informationen, die erst verfügbar sind, nachdem der Benutzer Posts oder Tags zu Bibsonomy hinzugefügt hat. Daher sind diese Features für diese Arbeit ungeeignet.

Zafarani et. al. [31] beschrieb einen Ansatz bösartige Benutzer mit einem minimum an Informationen zu identifizieren. Dazu benutzte sie lediglich den Benutzernamen. Diese nehmen an, dass bösartige Benutzer komplexe und vielfältige Benutzernamen wählen, um ihre Anonymität zu gewährleisten. Als ein Maß von Komplexität benutzen sie den Informationsgehalt des Benutzernamens. Als Maß für die Vielfältigkeit benutzen sie die Anzahl und den Anteil von Zahlen im Benutzernamen. Ein weiteres Feature aufbauend auf die erwünschte Anonymität der bösartigen Benutzer ist die

Entropy des Benutzernamens. Theoretisch, so argumentieren sie, sei ein Maximales Level an Anonymität gegeben, wenn ein Benutzername die maximale Entropy hat. Zusätzlich wurde die Entropy durch $log_2(n)$ normiert , wobei n die Anzahl der verschiedenen Buchstaben im Benutzernamen ist. Des Weiteren argumentieren sie, dass bösartige Benutzer ähnlich seien. Hierfür untersuchten sie Sprach- und Wortmuster im Benutzernamen. Zur Identifizierung von Sprachmustern wurden n-Gramme, die Anzahl und den Anteil an führenden Zahlen, die Anzahl der verschiedenen Zeichen, sowie Maximale Wiederholung von Bcustaben als Feature benutzt. Zur Erkennung von Wortmustern wurden Wörterbücher mit für bösartige Benutzer typischen Schlüsselwörtern benutzt. Mit diesen wurde das Aufkommen von solchen Schlüsselwörtern im Benutzernamen gezählt. Zudem benutzen sie Demographische Daten, wie Alter, Geschlecht, Sprache und Wissen.

Im Bereich der Sozialen Netzwerke wurden ebenfalls versucht Spammer zu klassifizieren. Zhu et. al. [33] klassifizierten Spammer mittel Matrix-Faktorisierung und Hu et. al. [15] anhand von Twitter-Posts. Diese Arbeiten benutzen jedoch Information, wie die Beziehungen der Benutzer untereinander, die direkt nach der Registrierung jedoch noch nicht zur Verfügung stehen und daher für diese Arbeit ungeeignet sind.

Große Online-Dineste, wie Google Mail, vertrauen heute längst nicht mehr auf ein korrekt eingegebenes Passwort bei der Authentifizierung [6]. Mittels Machine-Learning werden anhand mehrerer hunderter Features überprüft, ob ein Konto kompromittiert wurde. Ein Bereich in dem, unter anderem, Machine-Learning zu Identifizierung von Benutzern eingesetzt wird, ist Biometrie. Anhand der Art der Bedienung von Maus und Tastatur [12] [18] [32] [1] wird versucht einen Benutzer zu identifizieren. Feher et. all. [12] identifizierten Benutzer anhand dessen Mausverhaltens. Zum einen bezogen Sie Features auf Basis der Mausbewegungen selbst wie Geschwindigkeit, Beschleunigung, Krümmung, sowie Dauer und die zurückgelegte Distanz der Mausbewegungen mit ein. Zum anderen Features basierend auf dem Klick-Verhalten mit den Maustasten, wie z.B. die Zeit zwischen dem Drücken und Loslassen der Maustaste. Wesentliche Kriterien zur Analyse des Tippverhaltens ist die Dwell Time und die Flight Time [1]. Die Dwell Time gibt die Zeit zwischen drücken und Loslassen einer Taste an. Die Flight Time die Zeit zwischen Loslassen einer Taste und drücken der nächsten Taste.

Ein großer Teil der oben genannten Features wurden in diese Arbeit übernommen. Ihre Verwendung und Implementierung ist in Kaptitel 5 genauer beschrieben.

4 Datensatz

In der Vergangenheit wurden in Bibsonomy bereits Benutzer als Spammer und Nicht-Spammer klassifiziert. Dieses geschah zum einen manuell durch die Administratoren von Bibsonomy. Zum anderen wurden die Benutzer durch ein bereits implementiertes Spammer-Erkennungs-Framework [21] klassifiziert. Für die im folgenden dargestellten Daten gilt, dass nur solche Daten benutzt werden die einer manuellen Klassifizierung durch die Administratoren von Bibsonomy zugeordnet werden können, um zu vermeiden, dass der Klassifizierungsfehler des Frameworks diese Arbeit beeinflusst.

Die Entscheidung, ob ein Benutzer ein Spammer ist oder nicht, ist ein subjektiver Prozess. Ein Benutzer, der für eine Person ein normaler Benutzer ist, kann von einer anderen Person als Spammer angesehen werden. Wie Krause et. al. [21] bereits erwähnt wird Bibsonomy von mehreren Administratoren verwaltet. Auch die Klassifizierung eines Benutzers als Spammer oder Nicht-Spammer wird von unterschiedlichen Personen getätigt. Aus den oben genannten Gründen ist davon auszugehen, dass die verwendeten Daten eine gewisse Ungenauigkeit besitzten. Deshalb ist die Übertragung auf andere Bereiche als einem wissenschaftlich geprägten Umfeld wie Bibsonomy mit Vorsicht zu genießen. Menschen in einem anderen Kontext können Spammer anders auffassen, als die Administratoren von Bibsonomy.

In dieser Arbeit wurden Daten aus drei Quellen benutzt. Eine Teilmenge der Benutzer-Tabelle der Bibsonomy-Datenbank, die Apache-Log-Daten des Bibsonomy-Webservers und Interaktionen mit der Webseite, die während der Registrierung bei Bibsonomy, mittels Javascript aufgezeichnet wurden. In den folgenden Abschnitten werden die einzelnen Daten beschrieben, sowie ihre Erzeugung und Verwendung erläutert.

4.1 Bibsonomy Benutzer Tabelle

Ein erster Ansatz war es bereits in Bibsonomy vorhandene Daten zu benutzen. Dazu wurden die Daten von ausgewählten Attributen der Bibsonomy Benutzer-Tabelle exportiert. Zum einen sind dies die Angaben, die Benutzer bei der Registrierung macht. Ein Benutzername und eine E-Mai-Adresse muss der Benutzer bei der Registrierung angeben. Eine Homepage und ein Echtname sind freiwillige Angaben. Das Passwort wurde bewusst nicht benutzt. Auf das Passwort wurde, zusammen mit einem Salt,

Tab. 4.1.: Benutzer-Statistik Bibsonomy-Datenbank

	Gesamt	Homepage	Echtname	Früh. Reg.	Spät. Reg.
Spammer	180376	65295	131687	15 Nov 2006	10 Nov 2015
Nicht-Spammer	3614	736	1775	3 Dez 2006	8 Nov 2015

eine Hashfunktion angewendet, bevor dieses gespeichert wurde, und enthält somit keine verwertbaren Informationen mehr. Neben diesen Angaben wurden noch die, dem Benutzer zugeordnete IP-Adresse, das Datum und die Uhrzeit der Registrierung, sowie die Angabe, ob der Benutzer als Spammer klassifiziert wurde oder nicht, verwendet.

Tabelle 4.1 zeigt eine Statistik über die Benutzerzahlen, sowie die frühesten und spätesten Vorkommen von Benutzern, getrennt nach Spammer und Nicht-Spammer. Neben der Anzahl der Benutzer insgesamt, werden auch die Anzahl der Benutzer mit einer angegebenen Homepage bzw. Echtname aufgeführt. Es ist zu sehen, dass es wesentlich mehr Spammer in Bibsonomy gibt als Nicht-Spammer. Auf jeden Nicht-Spammer kommen ca. 50 Spammer. Zudem geben Spammer häufiger eine Homepage (36% der Spammer zu 20% der Nicht-Spammer) oder einen Echtnamen (73% der Spammer zu 49% der Nicht-Spammer) an, als Nicht-Spammer.

4.2 Interaktionen bei der Registerierung

In dem vorherigen Abschnitt wurden, unter anderem, Daten beschrieben, die der Benutzer bei der Registrierung angegeben hat. Neben den Daten selbst, kann auch betrachtet werden, wie die Daten in die Formulare bei der Registrierung eingegeben wurden. Diese Informationen wurden von Bibsonomy noch nicht gesammelt. Zu diesem Zweck wurde ein Eventlogger in Javascript implementiert und in die Registrierung von Bibsonomy eingebunden.

Abbildung 4.1 zeigt die Mausbewegung und die Mausklicks während der Registrierung bei Bibsonomy von 2 verschiedenen Benutzern. Die durchgängigen Linien zeigen den Verlauf der Maus. Die Rechtecke zeigen das drücken eines Maus-Buttons und die Kreise dessen Loslassen. Das Schwarz ausgefüllte Rechteck bezeichnet den Startpunkt der Aufzeichnung. Die Deckkraft der Farben beginnt zu beginn bei 10 % und steigert sich linear zur vergangenen Zeit hin zu 100% Deckkraft. Es ist zu sehen, dass die Art der Ausfüllung des Registrierung-Formulars von Bibsonomy sehr unterschiedlich sein kann.

Tabelle 4.2 zeigt die Benutzer-Statistik der Benutzer, dessen Verhalten während der Registrierung aufgezeichnet wurde. Das Verhältnis von Spammern zu Nicht-

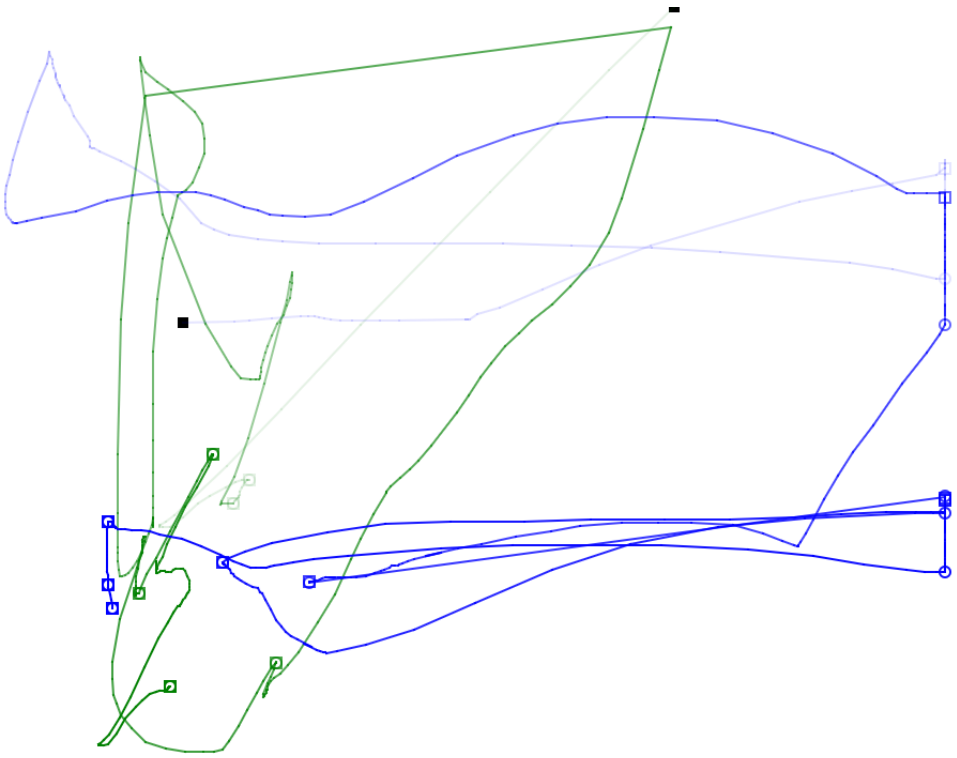


Abb. 4.1.: Mausbewegung und Mausklicks bei der Registrierung von Bibsonomy

Spammern liegt bei ca Elf zu Eins. Auch in diesen Daten tendieren Spammer häufiger dazu eine Homepage (74% zu 33%) und einen Echtnamen (79% zu 50%). Ca. 5% aller Benutzer, sowohl bei den Spammern, als auch bei den Nicht-Spammern, haben keine Daten an Bibsonomy übermittelt. Diese könnte zum einen dadurch bedingt sein, dass der Post direkt an Bibsonomy gesendet wurde. Zum anderen könnte auch JavaScript im Browser deaktiviert sein. Betrachtet man alle gesammelten Daten, also sowohl die bereits klassifizierten, als auch die noch nicht manuell klassifizierten, so ist der Anteil an solchen Benutzern, mit 25% deutlich höher. Das deutet darauf hin, dass dieser Datensatz zu einem gewissen Teil fehlerbehaftet ist. Es sei jedoch anzumerken, dass diese Tatsache die Bewertung der Features in Kapitel 5.6 nicht beeinflusst, da dort so oder so nur Benutzer mit übermittelten Daten betrachtet werden.

Tab. 4.2.: Benutzer-Statistik: Event-Log der Registrierung

	Gesamt	Homepage	Echtname	Früh. Reg.	Spät. Reg.
Spammer	1056	786	837	29 Jan 2016	21 Apr 2016
Nicht-Spammer	98	33	49	29 Jan 2016	21 Apr 2016

4.2.1 Event-Logger

Um alle Interaktionen des Benutzers zu erfassen wurden folgende Event-Typen mittels JQuerry Eventhandler erfasst: blur, focus, focusin, focusout, load, resize, scroll, unload, click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave, change, select, submit, keydown, keypress, keyup und error. Algorithmus 2 zeigt wie der Event-Logger initialisiert und die Listener gesetzt werden. Nach der Initialisierung des Loggers, wird überprüft, ob schon geloggte Daten im DOM-Dokument vorhanden sind. Dieses tritt immer dann auf, wenn der Benutzer Registrieren klickt und ein Fehler in den Eingabedaten vorliegt, wie z.B. eine ungültige E-Mail-Adresse. Anschließend werden diese Fehler in den Eingabedaten, sowie der Referer und User-Agent erfasst. Anschließend werden Listener für die oben genannten Event-Typen gesetzt. Letztlich wird dem Registrieren-Button ein Listener für den Submit gesetzt, sodass die Daten des Loggers in den DOM geschrieben werden und mit an Bibsonomy geschickt wird.

Algorithmus 2 Eventlogger registrieren

$logger \leftarrow Logger()$
if #registerUser.registrationLog exists in DOM **then**
    apply existing data to logger
end if
$logger.addErrors()$ ▷ Fehler in den Eingabedaten, z.B. Email-Format ungültig.
$logger.addReferer()$
$logger.addUserAgent()$
$logger.setListener(window)$ ▷ window: DOM-Dokument des aktuellen Fensters
$submit.listener \leftarrow$ set logger as JSON to #registerUser.registrationLog in DOM

Tab. 4.3.: Gespeicherte Daten im Log nach Event-Typ

Event-Typ	Daten
select	Zeitpunkt Δ, Selektionsstart, Selektionsende
resize	Zeitpunkt Δ, Neue Breite Δ, Neue Höhe Δ
scroll	Zeitpunkt Δ, Scroll-Position Δ
click*	Zeitpunkt Δ, X Δ, Y Δ
mouse*	Zeitpunkt Δ, X Δ, Y Δ
key*+	Zeitpunkt Δ, Key-Code, Alt, Ctrl, Shift
Sonstige	Zeitpunkt Δ

Δ Daten werden als Differenz gespeichert
* Alle Events, die mit vorangegangenen Prefix beginnen
+ Nur falls das Event-Ziel kein Passwort-Eingabefeld ist

Algorithmus 3 zeigt den Log-Vorgang eines Events. Zunächst werden Event-Typ und Event-Ziel extrahiert. Das Ziel ist die Id des Objektes im Dom von dem aus das

Event ausgelöst wurde. Ist keine Id für dieses Objekt vergeben, ist das Event-Ziel der leere String. Anschließend wird mit dem zuletzt verwendeten Events der gleichen Typ-Ziel Kombination die Zeitliche Differenz zwischen dem aktuellen und letzten Event gebildet. Je nach Event-Typ werden noch weitere Daten gespeichert. Diese sind in Tabelle 4.3 abgebildet.

Algorithmus 3 Event loggen

procedure LOG(event)
  $type \leftarrow getEventType(event)$ ▷ Strings werden durch Integer kodiert
  $target \leftarrow getEventTarget(event)$ ▷ Strings werden durch Integer kodiert
  $last \leftarrow getLastEvent(type, target)$ ▷ Default [0, 0, 0]
  $cur \leftarrow [event.time - getStartTime()]$
  $diff \leftarrow [cur[0] - last[0]]$
  if $type == TYPE$ **then**
    push data to cur and diff
  else if ... **then**
  end if
  $last \leftarrow cur$
  $getLoggedEvents(type, target).push(diff)$
end procedure

4.3 Apache-Log

Die Daten, die bei der Interaktion des Benutzers bei der Registrierung gesammelt werden, sind für die Daten aus Abschnitt 4.1 nicht vorhanden, da diese nur für Registrierungen ab der Implementierung in Bibsonomy erfasst werden. Um diese Daten mit weiteren Informationen anzureichern wurden die Apache-Log-Dateien analysiert. Das Log Schema von Bibsonomy enthält unter anderem neben dem

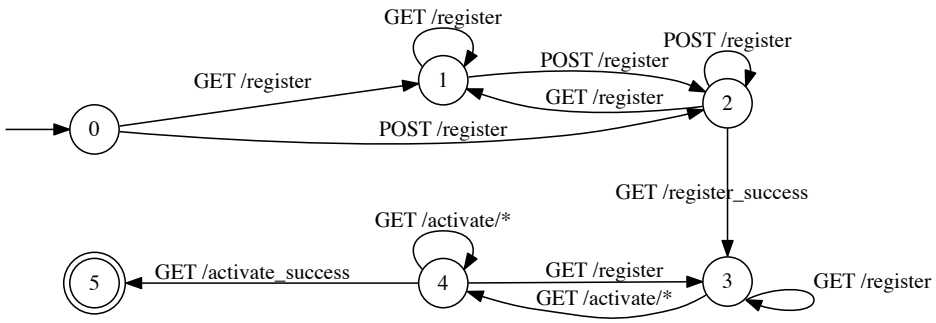


Abb. 4.2.: Automat zur Entscheidung einer gültigen Registrierung

Zeitstempel, Referer, User-Agent und Session-ID, auch die Art der Anfrage und die angefragte Seite selbst. Mit diesen Informationen ist es möglich den Start- und Endzeitpunkt einer Registrierung, sowie den Zeitpunkt der Aktivierung zu ermitteln. Zusätzlich kann der Referer des ersten Aufrufs der Registrierung und der User-Agent ermittelt werden. Zum User-Agent sei Anzumerken, dass nur die übermittelten Daten erfasst werden können. Ob die Angaben im User-Agent tatsächlich die richtigen sind, kann nicht überprüft werden.

Um diese Daten zu extrahieren werden die Log-Dateien monatlich ausgewertet. Zunächst werden nur solche Anfragen aus den Log-Dateien benutzt, die auf die Seiten register, register_success und activate, samt beliebigen Suffix, verweisen. Diese Anfragen werden dann anhand ihrer Session-Id in einzelne Listen aufgeteilt. Diese einzelnen Listen werden anschließend nach ihrem Zeitstempel von frühesten Zeitpunkt zu spätesten Zeitpunkt sortiert. Es wird angenommen, dass sich pro Session-Id nur ein Benutzer zur selben Zeit bei Bibsonomy registriert hat. Um aus diesen Listen von Anfragen eine gültige und eindeutige Registrierung zu extrahieren wurde ein modifizierter endlicher Automat erstellt. Die Modifikation besteht darin, dass der endliche Automat eine Teilmenge der Eingabe akzeptiert, sobald ein Endzustand erreicht wurde. Zudem hat dieser die Möglichkeit für gewisse Zustände Marker zu setzen. Somit ist es möglich zu wissen, welche Anfrage z.B. dazu geführt hat, dass die Aktivierung erfolgreich war. Der konkrete Automat ist in Abbildung 4.2 zu sehen. Dieser wurde nach folgenden Überlegungen aufgebaut: Zunächst hat der Benutzer noch keine der gefilterten Seiten aufgerufen (Zustand 0). Um sich zu registrieren ruft der Benutzer die Registrierung auf (GET auf register) und befindet sich auf dieser (Zustand 1). Anschließend gibt der Benutzer seine Daten ein und klickt auf registrieren (POST register). Der Server überprüft die Eingaben (Zustand 2) und leitet bei einem Fehler wieder auf Registrierung (GET register) um. Der Benutzer sieht die Registrierung erneut mit den entsprechenden Fehlermeldungen (Zustand 1). Werden die Eingaben aktzeptiert (GET register_success) hat sich der Benutzer erfolgreich bei Bibsonomy registriert (Zustand 3). Ebenso ist es möglich, dass der Benutzer die Webseite gar nicht erst aufgerufen hat und sich direkt per POST aufruft, z.B. durch ein entsprechende Skript, welches die Anmeldung automatisiert ausführt. Danach befindet sich der Benutzer direkt in Zustand 2. Nach der erfolgreichen Registrierung muss der Benutzer noch seinen Account aktivieren. Dazu klickt der Benutzer auf den Link in seiner E-Mail (GET activate*). Der Server überprüft den Aktivierungscode (Zustand 4) und zeigt einen Fehler an (Weiterhin Zustand 4) oder leitet bei Erfolg auf die Seite (GET activation_success) um, die dem Benutzer anzeigt, dass er seinen Account erfolgreich aktiviert hat (Zustand 5). Ist die Aktivierung nicht erfolgreich kann ein Benutzer eventuell noch mal die Registrierung angeklickt haben. Dieses ist in den Zuständen 3 und 4 beliebig oft möglich. Dies kann dadurch zustande kommen, dass der Benutzer nach der Registrierung oder erfolglosen Aktivierung unsicher ist und die Registrierung erneut aufruft, um dortige Hinweise zu lesen.

Anschließend müssen die gefundenen Registrierungen den Benutzern in Bibsonomy zugeordnet werden. Da Bibsonomy auf verschiedenen Server läuft und diese nicht vollständig synchronisiert sind, ist eine Zuordnung über die exakte Registrierungszeit nicht möglich. Dazu werden diese in Intervalle von Zeitpunkten aufgeteilt. Dazu werden die Registrierungszeiten (entsprechend ihrer Darstellung als Anzahl an Millisekunden nach dem 01.01.1970) durch eine entsprechende Intervallgröße geteilt und abgerundet. In diesem Fall wurde eine Intervallgröße von 1000 gewählt. Das führt dazu, dass alle Zeitpunkte, die in der selben Sekunde stattfinden zu einem Zeitpunkt zusammengefasst werden. Ein Zuordnung findet dann statt, wenn in einem Intervall genau ein Benutzer und genau eine Registrierung liegen und zusätzlich in dem vorherigen. Zusätzlich dürfen, sozusagen als Schutzabstand, vor und nach dem Intervall weder gefundene Registrierungen, noch registrierte Benutzer liegen, sodass die Wahrscheinlichkeit einer Falschzuordnung, Aufgrund von asynchronen Uhren auf den verschiedenen Servern gering gehalten werden kann.

Tab. 4.4.: Benutzer-Statistik: Apache-Log

	Gesamt	Homepage	Echtname	Früh. Reg.	Spät. Reg.
Spammer	11042	7014	7827	27 Mai 2010	10 Nov 2015
Nicht-Spammer	413	63	138	27 Mai 2010	21 Okt 2015

Tabelle 4.4 zeigt die Benutzer-Statistik der Extrahierten Benutzer au den Apache-Log-Dateien. Das Verhältnis von Spammern zu Nicht-Spammern liegt bei ca 1 zu 26. Auch in diesen Daten tendieren Spammer häufiger dazu eine Homepage (63% zu 15%) und einen Echtnamen (70% zu 33%) anzugeben.

5 Features

In diesem Kapitel werden die verwendeten Features dargestellt. Diese lassen sich in die Kategorien Sprach-basierte, Umwelt-basierte, Tastatur-basierte, Interaktions-basierende und auf Log-Daten basierende Features unterteilen.

Es wurde zwischen zwei Arten von Features unterschieden (Siehe Abbildung 5.1). Zum einen numerische Features, die Eigenschaften eines Benutzers auf eine Natürliche bzw. Reelle Zahl abbilden, sowie nominale Features, die Eigenschaften des Benutzers auf Werte, eines endlichen, festgelegten, Wertebereichs abbilden. Boolean Features stellen eine Spezialform der nominalen Features dar, welche den Wertebereich {0, 1} benutzen. Diese Abbildung wird bei numerischen Features durch das Implementieren des Interfaces $Function < User, Double >$ und bei nominalen Features durch Implementieren des Interfaces $Function < User, String >$ umgesetzt. Die Resultierenden Strings bei nominalen Features müssen dabei innerhalb des festgelegten Werftenbereichs sein ($possibleValues$). Das Attribut eines Features bildet den Wertebereich des Features auf eine Weka-Kompatible Struktur ab. Eine Besonderheit stellen Features dar, die die Gesamtheit der Benutzer benötigen. Diese können über einen Boolean-Wert signalisieren, dass diese die Gesamtheit benötigen. Ist dieser Wert gesetzt, wird bei der Initialisierung des Features für diese Feature, die Methode $setUser$ aufgerufen, sodass diese entsprechende Vorberechnungen ausführen können.

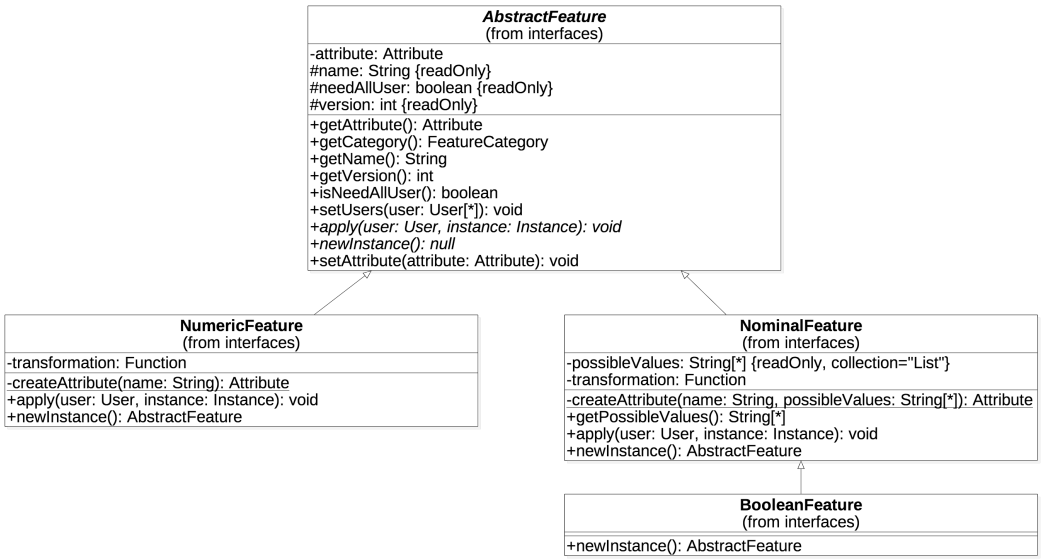
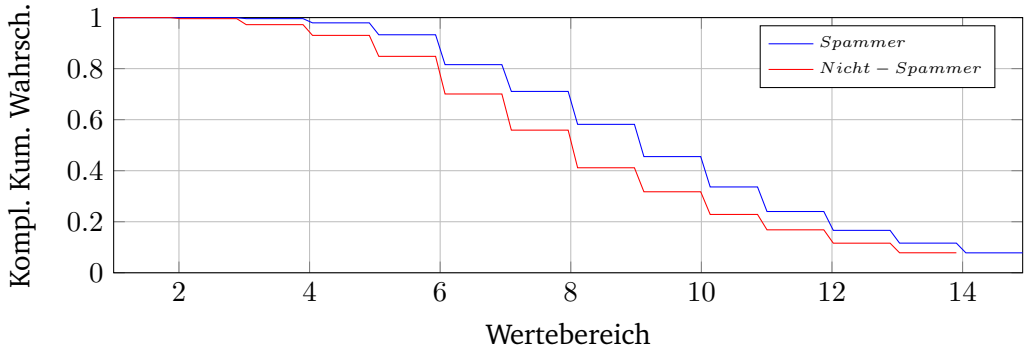


Abb. 5.1.: Klassendiagramm Features

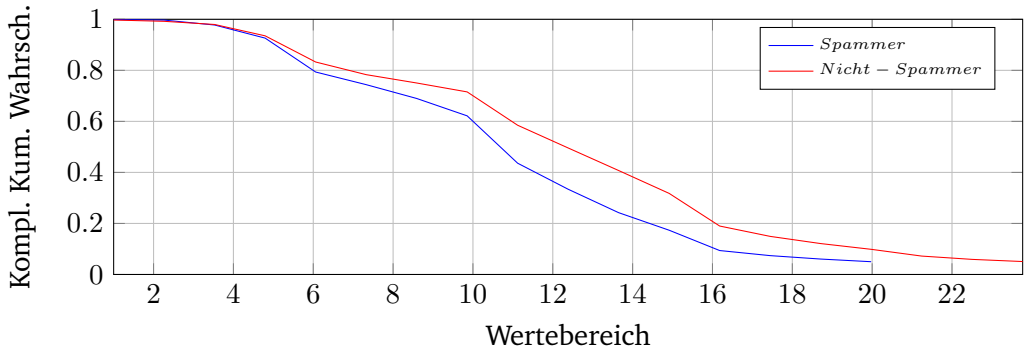
Zur Veranschaulichung werden Diagramme von Empirische Verteilungsfunktionen und diskreten Werte-Verteilungen ausgewählter Features benutzt. Eine Übersicht aller verwendeten Features und die Diagramme der Verteilungen aller Features sind im Anhang zu finden. Um zu entscheiden, ob sich die Empirische Verteilungsfunktionen der Spammer und NIcht-Spammer in einem Feature unterscheiden, wurde der Kolmogorow-Smirnow-Test mit einem Signifikanzniveau von 0,1% verwendet.

5.1 Sprach-basierte Features

Dieser Abschnitt beschreibt Features, die die Angaben des Benutzers sprachlichen nach Auffälligkeiten und Mustern untersuchen. Grundlegende Informationen für diese Features liefern dabei der Benutzername, Echtname, sowie die E-Mail-Adresse und Homepage. Sofern nicht gesondert angegeben werden alle Meta-Features in diesem Abschnitt auf die vorherig genannten Eigenschaften des Benutzers angewendet. Alle verwendeten Diagramme in diesem Abschnitt wurden mit den Daten aus Kapitel 4.1 erzeugt.



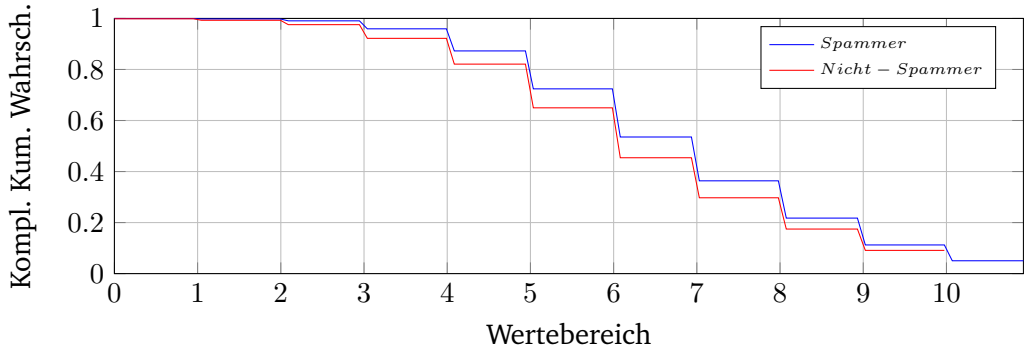
(a)nameLength



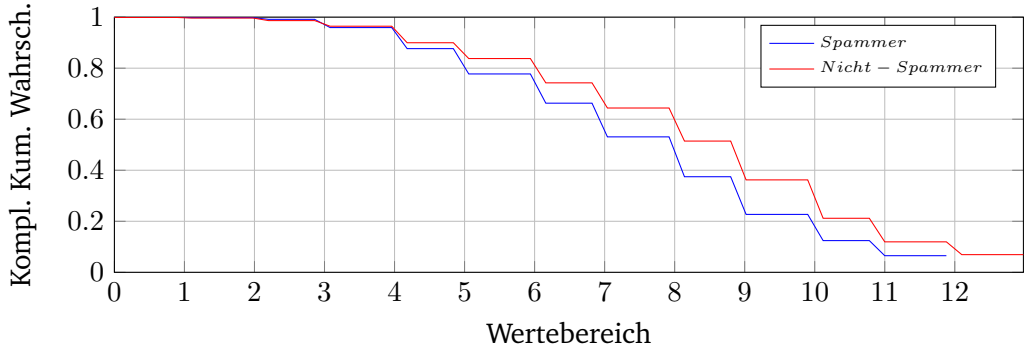
(b)realnameLength

Abb. 5.2.: Empirische Verteilungsfunktionen im Meta-Features $length$

Ein sehr simpler erster Ansatz ist die Länge eines Wortes, zusammengefasst im Meta-Feature $length$. Krause et. al. [21] verwendeten diesen Ansatz bereits erfolgreich zur Klassifizierung von Benutzern. Angewendet auf den vorhandenen Datensatz zeigen sich Unterschiede in der Länge der Eingaben zwischen Spammern und Nicht-Spammern bei dem Benutzernamen, der E-Mail-Adresse und dem Echtnamen. Die Länge der Homepage unterscheidet sich zwischen Spammern und Nicht-Spammern nicht. Spammer benutzen marginal längere E-Mail-Adressen. Bei den Echtnamen (Abbildung A.4c) ist dieser Unterschied deutlicher zu erkennen. Nicht-Spammer hingegen benutzen längere Echtnamen (Abbildung A.4d).



(a)nameNumUniqueAlphLetters



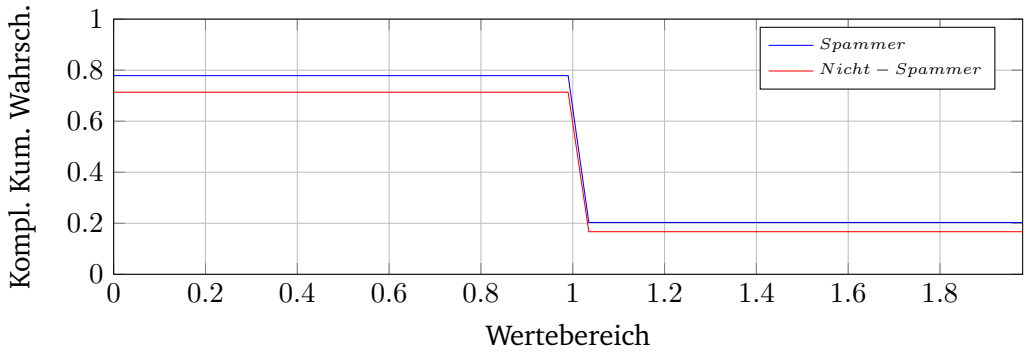
(b)realnameNumUniqueAlphLetters

Abb. 5.3.: Empirische Verteilungsfunktionen im Meta-Feature $numUniqueAlphLetters$

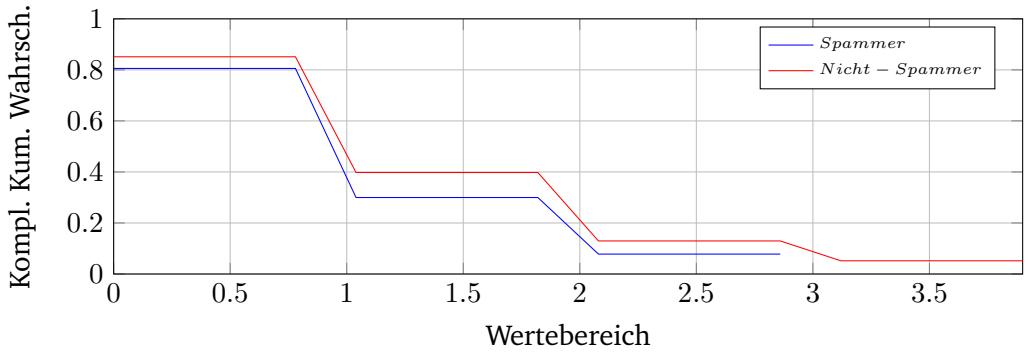
Ausgehend von der Anzahl Zeichen eines Wortes, ist ein weiteres Meta-Feature die Anzahl der verschiedenen Zeichen, genannt $NumUniqueAlphLetters$. Zafarani et. al. [31] benutzten dieses Feature, da ein Teil von bösartigen Benutzern besonders Effizient seien. Als Begründung führen Sie an, dass daran interessiert sind ihre Aktionen besonders häufig, schnell und in großen Umfang ausgeführt werden. Ausgegangen davon, dass Spammer aus Effizienz gründen möglichst wenig verschiedene Buchstaben benutzen, so werden diese auch nicht viele Buchstaben wiederholen

[31]. Das Meta-Feature $numMaxTimesLetterRep$ gibt die maximale Anzahl an Wiederholungen von Buchstaben in einem Wort an.

Abbildung 5.4 und 5.3 zeigen, dass sich die Werte der Feature $nameNumUniqueAlphLetters$ und $realnameNumUniqueAlphLetters$, bzw. $nameNumMaxTimesLetterRep$ und $realnameNumMaxTimesLetterRep$, für Benutzername und Echtnamen zwischen Spammern und Nicht-Spammer unterscheiden. Bei der Homepage gibt es weder bei bei Anzahl der verschiedenen Buchstaben, noch bei der maximalen Anzahl an Wiederholungen, signifikante Unterschiede. Bei der E-Mail-Adresse gibt es zudem bei der maximalen Anzahl an Wiederholungen keine signifikanten Unterschiede. Spammer tendieren dazu im Benutzernamen mehr verschiedene Buchstaben und häufiger Wiederholungen als Nicht-Spammer zu benutzen. Nicht Spammer tendieren hingegen bei den Echtnamen mehr verschiedene Buchstaben und häufiger Wiederholungen als Spammer zu benutzen.



(a)nameNumMaxTimesLetterRep

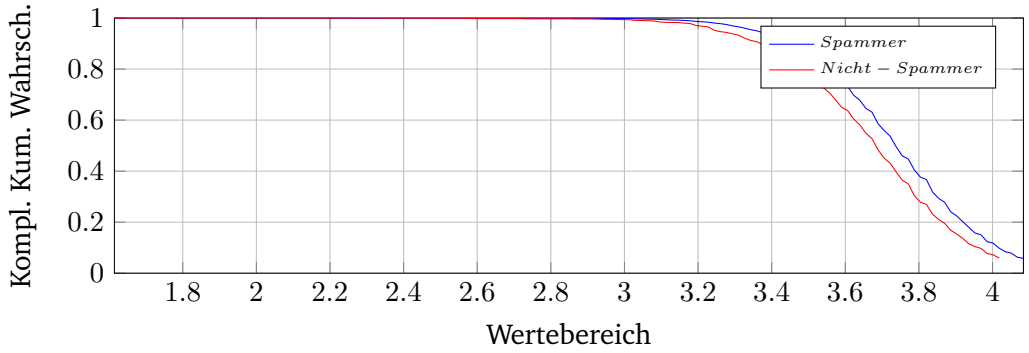


(b)realnameNumMaxTimesLetterRep

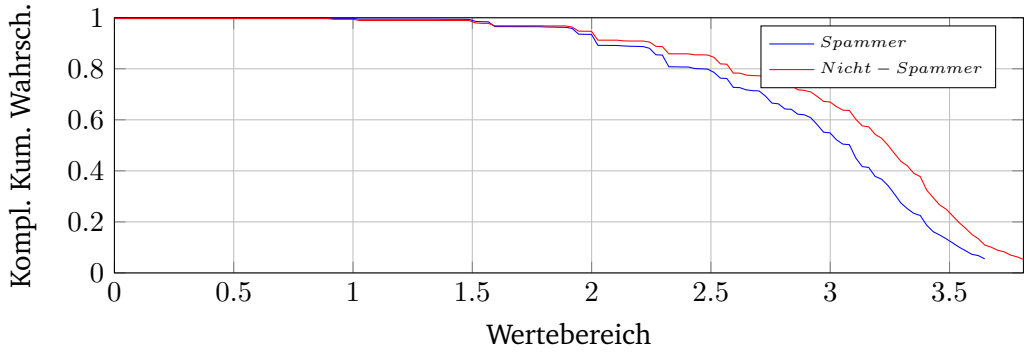
Abb. 5.4.: Empirische Verteilungsfunktionen im Meta-Feature $numMaxTimesLetterRep$

Ein weiterer Aspekt von Wörtern ist dessen Komplexität. Spammer möchten anonym bleiben, um bei ihren Tätigkeiten nicht erkannt zu werden [31]. Eine höchstmögliche Anonymität gewehrt eine Angabe dann, wenn seine Entropy am höchsten ist [31]. Das Meta-Feature $entropy$ gibt die Entropy eines Wortes nach Shannon an. Diese ist

definiert durch $H = -\sum_i p_i \log_b p_i$ [25], wobei p_i die Auftretenswahrscheinlichkeit des i-ten Buchstaben in der Menge der Buchstaben, in der Eingabe, ist. Die Entropy der Homepage unterscheidet sich nicht zwischen Spammern und Nicht-Spammern. Bei dem Benutzernamen gibt es deutlichere Unterschiede. Spammer tendieren zu einer höheren Entropy in dem Benutzernamen und der E-Mail-Adresse (Abbildung 5.5 a). Nicht-Spammer tendieren zu einer höheren Entropy beim Echtnamen (Abbildung 5.5 b).



(a)emailEntropy

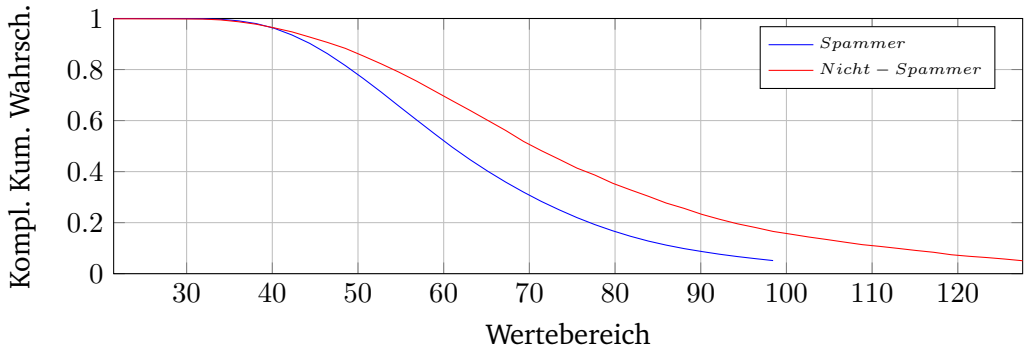


(b)realnameEntropy

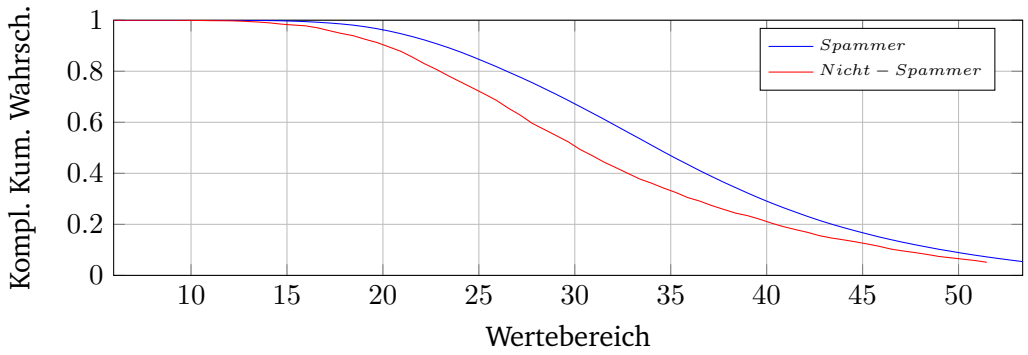
Abb. 5.5.: Empirische Verteilungsfunktionen im Meta-Feature *entropy*

Eine weitere Art von Komplexität ist die Kolmogorov-Komplexität. Zafarani et. al. [31] haben gezeigt, dass die Kolmogorov-Komplexität durch den erwarteten Informationsgehalt der Wörter in dessen Raum berechnet werden kann. Die Wahrscheinlichkeit eines Wortes wird dabei wie folgt berechnet: Gegeben sei ein Wort u, so ist dessen Auftretenswahrscheinlichkeit $P(u) = \prod_{i=1}^{n} p(c_i|c_1 \dots c_{i-1})$. Um diese Wahrscheinlichkeiten zu berechnen, wurden diese über Bigrams auf Zeichenebene approximiert. Dadurch wird die Wahrscheinlichkeit eines Wortes u zu $P(u) \approx \prod_{i=1}^{n} p(c_i|c_{i-1})$. Die bedingten Wahrscheinlichkeiten der einzelnen Zeichen werden dabei im vorhinein auf Basis aller Eingaben der Benutzer, des jeweiligen Feldes, berechnet. So werden für den Benutzernamen die bedingten Wahrscheinlichkeiten der einzelnen Zeichen

aus den Benutzernamen aller Benutzer berechnet. Um den Anfang und das Ende eines Wortes zu beschreiben, werden diesem ein Startsysmbol vorangestellt und ein Endsymbol nachgestellt. Der Informationsgehalt der Homepage unterscheidet sich nicht signifikant zwischen Spammern und Nicht-Spammern. Es ist zu erkennen, dass Spammer zu einem größeren Informationsgehalt im Benutzernamen und der E-Mail-Adresse (Abbildung 5.6 a) tendieren. Bei dem Echtnamen (Abbildung 5.6 b) tendieren Nicht-Spammer zu einem größeren Informationsgehalt.



(a)emailInfoSupr

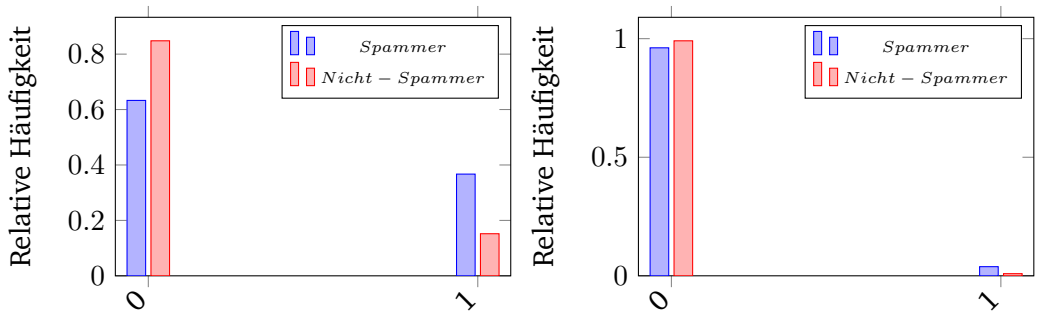


(b)nameInfoSupr

Abb. 5.6.: Empirische Verteilungsfunktionen im Meta-Features $infoSupr$

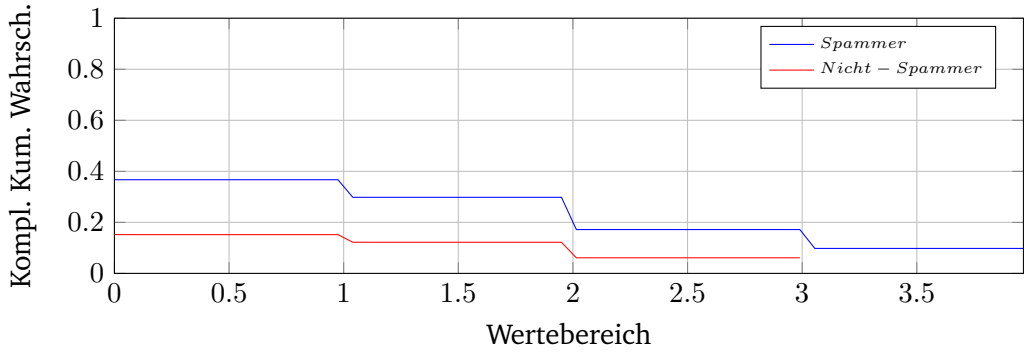
Eine weitere Auffälligkeit sind Zahlen in den Angaben von Benutzern. Spammer tendieren dazu öfters Zahlen zu benutzen als nicht Spammer [21]. Zur Erfassung dieser Beobachtung ist das einfachste Meta-Feature, jenes welches angibt, ob eine Zahl in einem String vorhanden ist. Dieses Meta-Feature wird $digit$ genannt. Abbildung 5.7 zeigt, dass es einen deutlichen Unterschied bei der Benutzung von Zahlen zwischen Spammern und Nicht-Spammern gibt. In dem Benutzernamen und der E-Mail-Adrese benutzt ca. jeder Fünfte Nicht-Spammer eine Zahl, während bereits ca. jeder dritte Spammer eine Zahl benutzt. Bei dem Echtnamen und der Homepage benutzen sehr wenige Benutzer, sowohl Spammer, wie Nicht-Spammer, Zahlen, sodass ein Unterschied insgesamt nur sehr gering ausfällt. Dieses liegt

wahrscheinlich an der Natur von Namen und einer Homepage. Es ist nicht üblich, dass Zahlen in Namen vorkommen. Auch bei einer Homepage ist die Anzahl von Zahlen eher gering.

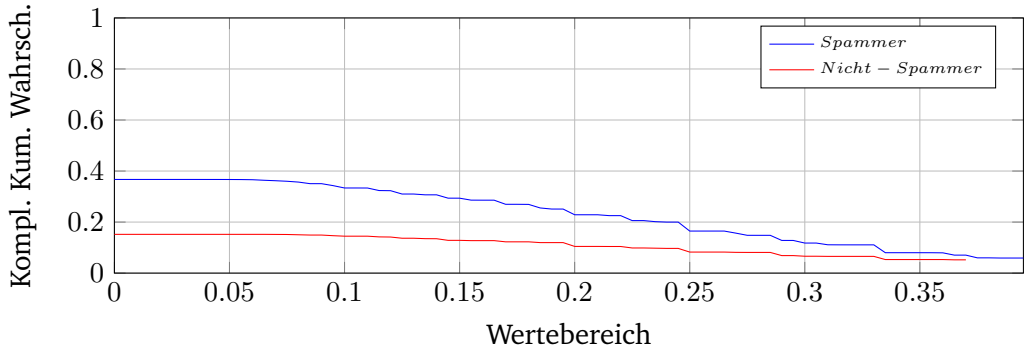


(a)nameDigit **(b)**realnameDigit

Abb. 5.7.: Werte-Verteilungen im Meta-Feature *digit*



(a)nameNumDigit



(b)namePropDigit

Abb. 5.8.: Empirische Verteilungsfunktionen im Meta-Features *numDigit* & *propDigit*

Die Meta-Feature *numDigit* und *propDigit* stellen eine genauere Betrachtung bei der Benutzung von Zahlen dar. *numDigit* beschreibt die Anzahl an Zahlen in einem String, *propDigit* die Anzahl an Zahlen in einem String relativ zu seiner

Gesamtlänge. Es ist zu beobachten, dass Spammer im Benutzernamen und in der Email-Adresse dazu tendieren mehr Zahlen zu verwenden (Abbildung 5.8). Sowohl absolut ($numDigit$), als auch relativ zur Länge des Wortes ($propDigit$). Bei der Homepage und dem Echtnamen ist zwischen Spammer und Nicht-Spammer kein signifikanter Unterschied festzustellen. Das liegt vor allem daran, dass in diesen die Anzahl der Zahlen sowieso sehr gering ist. Zafarani et. al. [31] beobachteten, dass Spammer dazu tendieren Benutzernamen mit Zahlen starten zu lassen. Diese Beobachtung wird durch das Meta-Feature $numStartDigit$ beschrieben, welches angibt wieviele führende Zahlen ein String hat. Angewendet auf unseren Datensatz kann diese Beobachtung jedoch nicht bestätigt werden. In unserem Datensatz benutzen 0,51 % aller Spammer mindestens eine führende Zahl. Bei den Nicht-Spammern sind es 0,43 %. In dem Kontext von Webspam und Bibsonomy scheint dieses Beobachtung keine Relevanz zu haben.

Krause et. al. [21] betrachteten, ob der Echtname aus Zwei oder Drei Teilen besteht. Diese wurden zu dem Feature $realnameParts$ zusammengefasst, welches angibt aus wie vielen Teilen der Echtname besteht, wenn dieser nach Leerzeichen aufgeteilt wird. Abbildung A.10a zeigt, dass Nicht-Spammer zu Echtnamen aus mehr Teilen tendieren, als Spammmer.

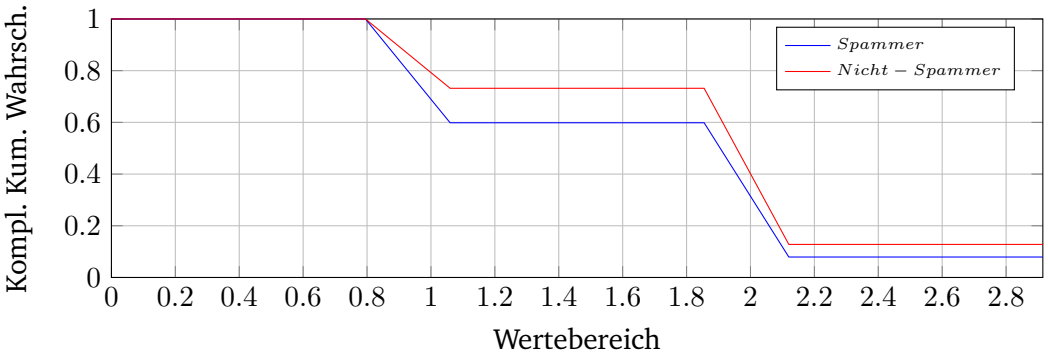
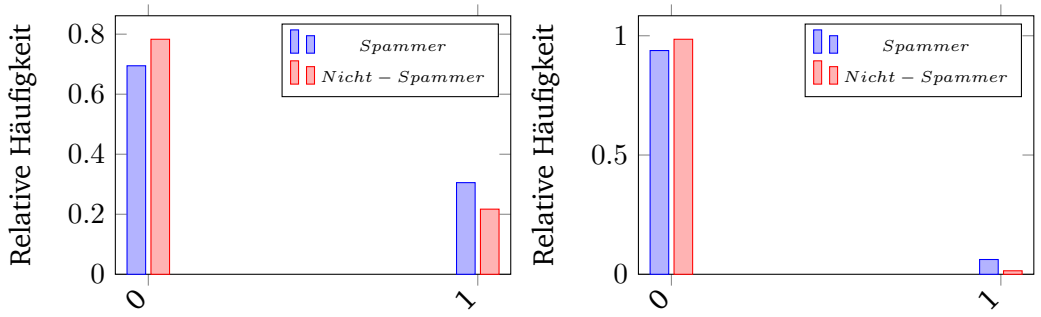


Abb. 5.9.: CCFD-Verteilung des Features $realnameParts$

Bei der Betrachtung der Daten ist aufgefallen, dass Spammer öfters den gleichen Benutzernamen und lokalen Teil der Email-Adresse , sowie Echtnamen, benutzen. Das Feature $nameEqualMail$ prüft, ob der Benutzername und der lokale Teil der E-Mail-Adresse gleich sind. Das Feature $nameEqualMailEqualRealnname$ prüft zusätzlich, ob auch der Echtname gleich dem Benutzernamen und dem lokalen Teil der E-Mail-Adresse ist. Abbildung A.5 zeigt, dass diese Beobachtung tatsächlich bei Spammern häufiger vorkommt.



(a) nameEqualMail **(b)** nameEqualMailEqualRealnname

Abb. 5.10.: Werte-Verteilungen der Feature $nameEqualMail$ & $nameEqualMailEqualRealnname$

Die in diesem Unterkapitel dargestellten Features haben Sprachliche Unterschiede in den Angaben der Spammer und Nicht-Spammer aufgezeigt. Auffällig dabei ist, dass sich das Verhalten bei Benutzernamen und Echtnamen zum Teil umgekehrt verhält. Spammer tendieren beim Benutzernamen und der E-Mail zu längeren Wörtern, mehr verschiedene Buchstaben und wiederholen diese häufiger. Bei dem Echtnamen ist dieses Verhalten genau umgekehrt. Hier tendieren Nicht-Spammer zu längeren Wörtern, mehr verschiedene Buchstaben und wiederholen diese häufiger. Bei der Homepage zeigen sich keine großen Unterschiede, unabhängig vom Feature.

5.2 Umwelt-basierte Features

Dieser Abschnitt beschreibt Features, die den Spammer aufgrund von Eigenschaften seiner Umwelt beschreiben, wie z.B. seinen Standort. Die nachfolgenden Features basieren auf der IP-Adresse, dem Registrierungsdatum und der E-Mail-Adresse des Benutzers.

(a) uniMail

Abb. 5.11.: Werte-Verteilungen des Features $uniMail$

Bibsonomy wird zu einem großen Teil von Benutzern aus dem akademischen Umfeld benutzt. Es ist anzunehmen, dass vielen von diesen Benutzern eine E-Mail-Adresse einer Hochschule benutzen. Außerdem ist anzunehmen, dass Universitäts-E-Mail-Adressen an berechtigte Personen, wie z.B. Studenten oder Mitarbeiter der Universität, ausgegeben werden. Diese Annahmen führen dazu, dass eine Person, die eine solche E-Mail-Adresse bei der Registrierung angibt, einer Universität zugeordnet werden kann. Das Feature $uniMail$ knüpft an diesen Gedanken an. Dieses gibt an, ob eine E-Mail-Adresse einer Hochschule zuzuordnen ist. Um zu überprüfen, ob die Top-Domain einer E-Mail-Adresse zu einer Universität gehört, wurde eine auf Github veröffentlichte Liste von Universitäten Weltweit [1] benutzt. Die Top-Domain der E-Mail-Adresse wird extrahiert. Ist diese Top-Domain in der Liste der Universitäten enthalten, so wird die E-Mail-Adresse als einer Universität zugehörend angenommen. Das eben beschrieben Verfahren ist nicht perfekt. Zum einen kann die verwendete Liste unvollständig sein. Zum anderen hält sich nicht jedes Institut einer Universität an das Muster <Benutzer>@<Instituts-Kürzel>.<Universitäta-Domain>. Außerdem können aus verschiedenen Gründen (Sicherheitslücken am E-Mail-Server, Phishing, ...) Spammer Zugang zu solch einer E-Mail-Adresse bekommen und für die Registrierung benutzt haben. Solche Probleme sollten jedoch eher von seltener Natur sein. Abbildung A.11c zeigt die Verteilung von Spammern und Nicht-Spammern mit einer Universitäts-Email-Adresse. Während 18,6% aller Nicht-Spammer eine Universitäts-E-Mail-Adresse haben, haben nur 0,7% aller Spammer eine.

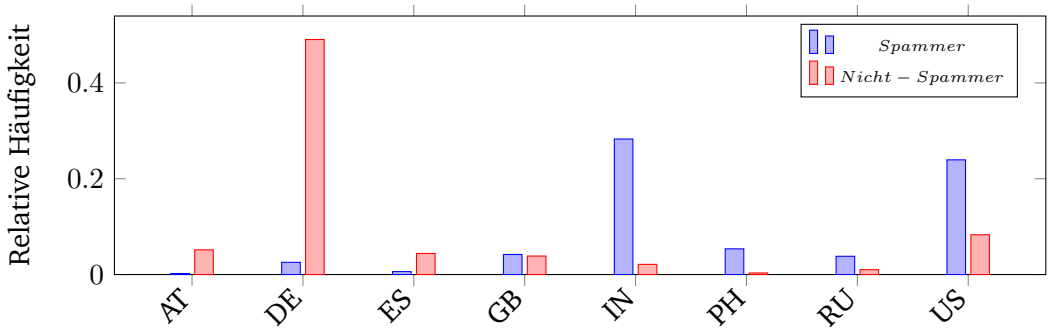
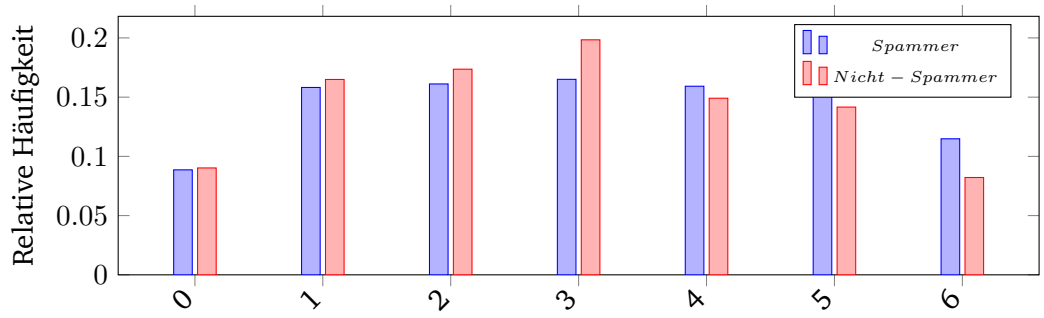


Abb. 5.12.: Top-5 Ursprungsländer der Spammer vereint mit Top-5 Ursprungsländer der Nicht-Spammer

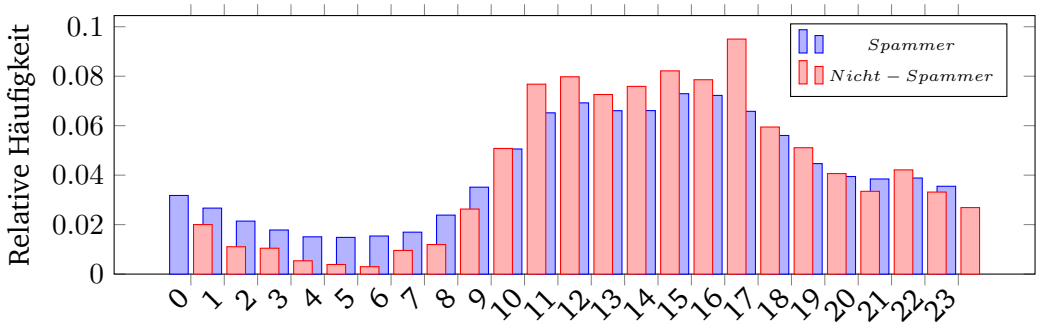
Analysen haben gezeigt [16], dass bestimmte Länder dazu tendieren mehr Spam zu produzieren als andere Länder. Aus diesem Grund wurde das Feature $country$ implementiert. Dieses Ordnet der IP-Adresse des Benutzers ihr entsprechendes Ursprungsland zu. Dazu wird die kostenlose IP-zu-Geo-Datenbank GeoLite2 von Maxmind [2] benutzt. Die Werte, die dieses Feature produziert, sind Alpha-2 Ländercodes nach ISO-3166. Abbildung 5.12 zeigt die Verteilung der Registrierungen aus den Top-5 Ursprungsländer der Spammer vereint mit den Top-5 Ursprungsländern der

[1] https://github.com/Hipo/university-domains-list
[2] https://dev.maxmind.com/geoip/geoip2/geolite2/

Nicht-Spammer. Es ist zu sehen, dass die meisten Nicht-Spammer aus Deutschland kommen. Während die meisten Spammer aus Indonesien kommen. In beiden Fällen hat die Komplement-Klasse nur einen sehr geringen Anteil in diesem Land.



(a)regDay



(b)regHour

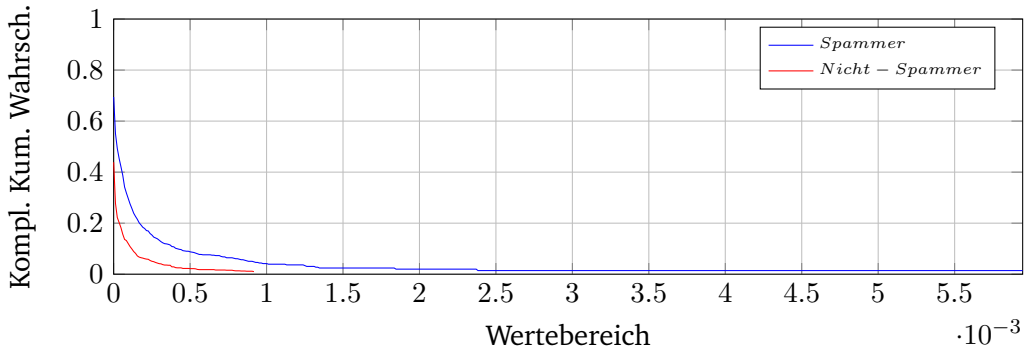
Abb. 5.13.: Werte-Verteilungen der Feature $regDay$ & $regHour$

Zusätzlich zu dem Land aus dem die Registrierung stattgefunden hat, wurde auch die Stunde ($regHour$) und der Tag ($regHour$) der Registrierung in der lokalen Zeitzone des Benutzers erfasst. Die Umrechung der Server-Zeit in die lokale Zeit des Benutzers erfolgte approximiert über die IP-Adresse des Benutzers. Beide Features sind als Nominale Feature mit Werten von 0 - 23 bzw. 0 -6, für Sontag bis Samstag, implementiert. Abbildung 5.13 zeigt dessen Werte-Verteilungen. Es ist zu sehen, dass sowohl Spammer, als auch Nicht-Spammer, die meisten Registrierungen zwischen 10 und 22 Uhr haben. Die Spammer haben jedoch einen deutlich höheren konstanten Anteil an Registrierungen den ganzen Tag über. Eine mögliche Erklärung könnte sein, dass dieser Anteil durch eine automatisierte Anmeldung zustande kommt. Ein Indiz hierfür ist, dass in den in Kapitel 4.2 beschrieben Daten, Benutzer vorhanden sind, dessen Post zur Registrierung keine Interaktions-Daten enthielt. Dieses lässt sich dadurch erklären, dass die Registrierung nicht über die Webseite erfolgte, sondern über ein Tool, das den Post direkt an den Bibsonomy-Server geschickt hat. Bei dem Tag der Registrierung ist zu erkennen, dass die Registrierungen der Nicht-Spammer zum Mittwoch hin zunimmt und zum Sonntag hin wieder abnimmt. Bei den Spammer, ist diese Verhalten wesentlich weniger stark ausgeprägt. Montag bis

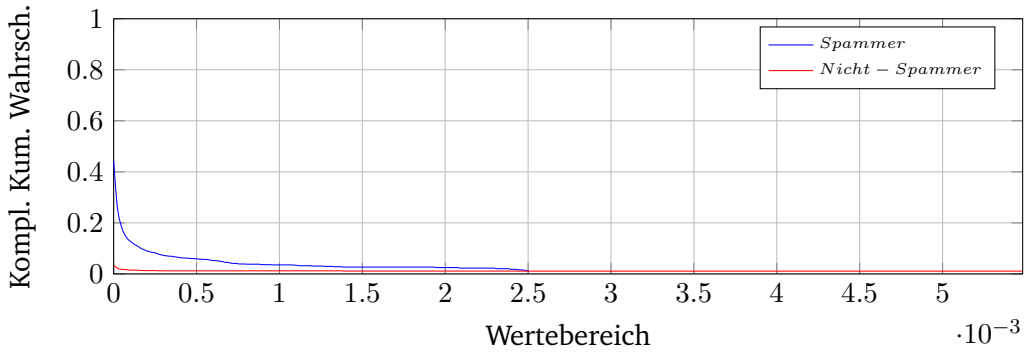
Freitag haben fast den selben Anteil an Registrierungen. Auch diese Beobachtung könnte für eine automatisierte Registrierung sprechen.

5.3 Populations-basierte Features

Dieser Abschnitt beschreibt Feature, die Eigenschaften eines Benutzers bezogen auf die Gesamtheit der Benutzer benutzt. Hierzu werden Eigenschaften gezählt ($count$) und in Relation ($ratio$) gesetzt. Da diese Feature die Gesamtheit der Benutzer einbeziehen, muss diesen diese zur Verfügung stehen. Daher werden diesen Feature beim erzeugen der Features für den Klassifikator die Gesamtheit der Benutzer übergeben und entsprechende Kennwerte vor ab berechnen.



(a)lastnameCount

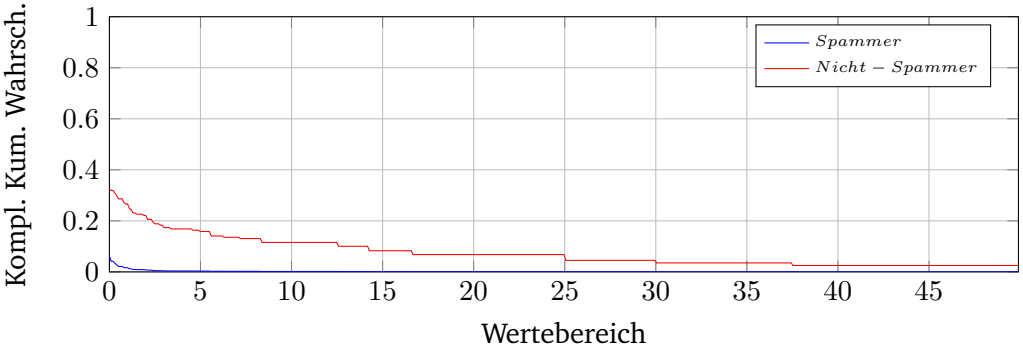


(b)spamIP

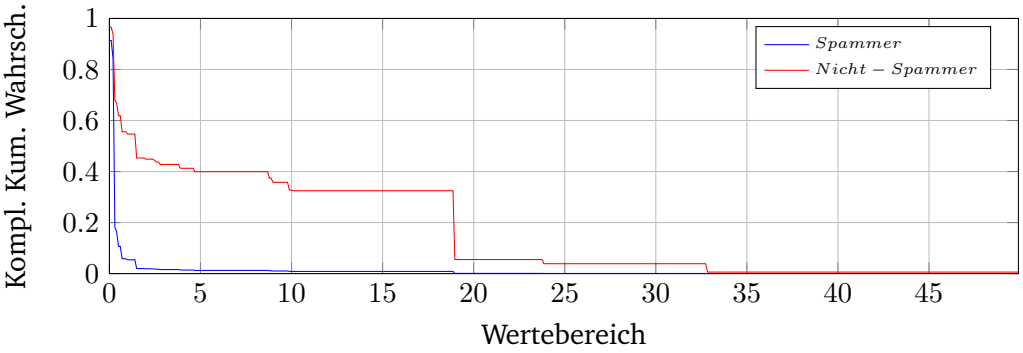
Abb. 5.14.: Empirische Verteilungsfunktionender Features $lastnameCount$ & $spamIP$ im Meta-Feature $count$

Ausgehend von der Annahme, dass ein Teil der Spammer automatisierte Prozesse zur Registrierung benutzen, ist es naheliegend, dass diese auf einen endlichen Vorrat von Werten zur Ausfüllung der Felder bei der Registrierung zugreifen. Es ist also wahrscheinlich, dass sich diese Werte mehrfach in den Benutzerdaten von Bibsonomy auftauchen. Für die folgenden Meta-Features wurde angenommen, dass

ein Wertevorat kombiniert mit Zahlen und Sonderzeichen benutzt wird. Um die Angaben des Benutzers von diesen Kombinationen loszulösen, wurden diese normalisiert. Normalisiert bedeutet in diesem Fall, dass die Eingaben zur Kleinschreibung umgewandelt und alle nicht alphabetischen Zeichen entfernt wurden.



(a)lastnameRatio



(b)tldRatio

Abb. 5.15.: Empirische Verteilungsfunktionen der Features $lastnameRatio$ & $tldRatio$ im Meta-Feature $ratio$

Das Meta-Feature $count$ zählt das relative Vorkommen von gemeinsamen normalisierten Eingaben mit anderen Benutzern in der Gesamtheit der Benutzer. Haben also von fünf Benutzern drei die gleichen normalisierten Eingaben, so hat diese Eingabe einen Wert von Ein-Halb, da zwei von vier anderen Benutzern diesen Wert haben. Werden diesem die Gesamtheit der Benutzer übergeben, zählt es die vorkommen der normalisierten Eingaben in einer Map mit. Über diese Map kann anschließend das relative Aufkommen zur normalisierten Eingabe zurückgegeben werden. Angewendet wird dieses Meta-Feature auf den Benutzernamen, dem lokalen Teil der E-Mail-Adresse, dem Echtnamen, dem Vornamen, dem Nachnamen, sowie der Domain und Top-Level-Domain der Homepage. Der Vornamen und Nachname ergibt sich dadurch, dass der Echtname nach Leerzeichen aufgeteilt wird. Der erste Teil des Echtnamens ist der Vorname und der letzte Teil der Nachname. Gibt es im Echtnamen keine Leerzeichen, wird Vorname und Nachname als fehlender Wert gewertet. Eine Beson-

derheit ist das Feature $spamIP$, welches das relative Vorkommen von Spammer mit der IP-Adresse des Benutzers zählt. Bis auf bei der Domain sind bei allen Features im Meta-Feature $count$ signifikante Unterschiede zwischen den Verteilungen von Spammern und Nicht-Spammern vorhanden. Sichtbare Unterschiede sind dabei vor allem bei dem Vor- und Nachnamen, dem Namen, der Top-Level-Domain und den IP-Adressen von Spammern zu erkennen. Hier tendieren Spammer zu einen höheren Anteil an gemeinsamen normalisierten Eingaben. Abbildung 5.14 zeigt, dass bei den Nachnamen und den gemeinsamen IP-Adresse mit Spammern ein deutlicher Unterschied in der Anzahl an gemeinsamen normalisierten Werten vorhanden ist.

Das Zählen von normalisierten Eingaben erfasst, wieviele Andere diese Eingabe teilen. Es erfasst jedoch nicht, ob eine solche Eingabe eher von Spammern oder Nicht-Spammern gemeinsam benutzt wird. Das Meta-Feature $ratio$ erfasst das Verhältnis der relativen Häufigkeiten von gemeinsamen normalisierten Eingaben zwischen Spammern und Nicht-Spammern. Abbildung 5.15 zeigt, dass es deutliche Unterschiede in den Verteilungsfunktionen zwischen Spammern und Nicht-Spammern gibt. Das Feature $lastnameRatio$ zeigt jedoch auch, dass ca. 70% der Nicht-Spammer einen Wert von Null haben. Diese lassen sich also nicht von ca 95 % der Spammer unterscheiden.

5.4 Tastatur-basierte Features

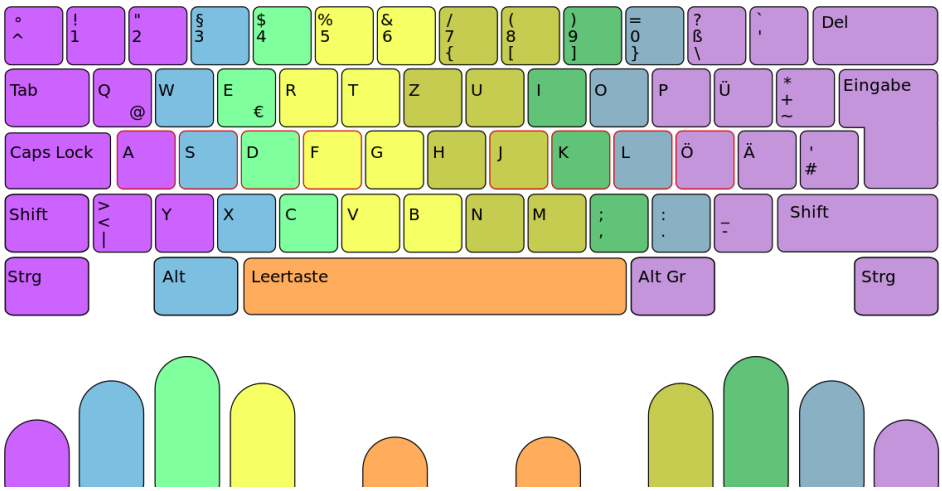
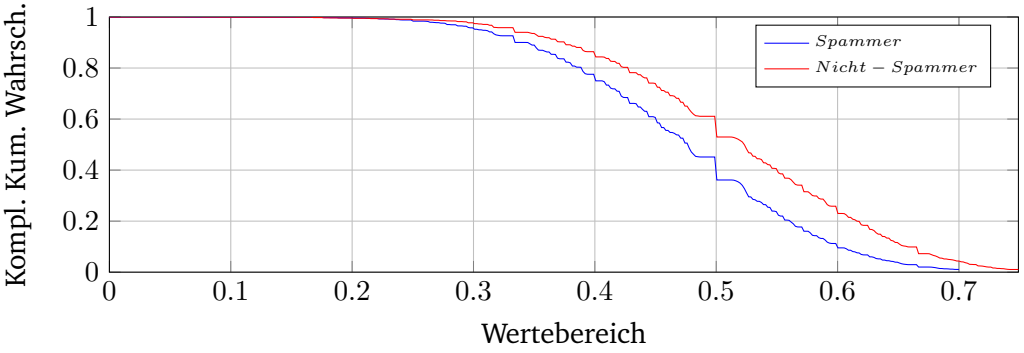


Abb. 5.16.: Zuordnung der Tasten im Zehfingersystem. Quelle: [@29]

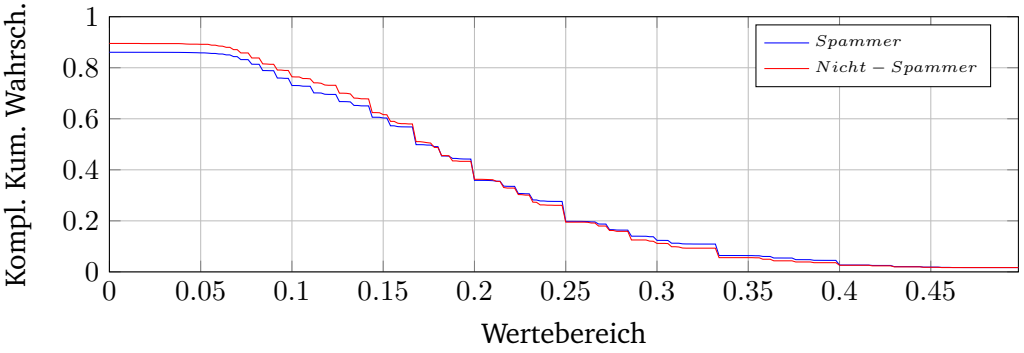
Zafarani et. al. [31] argumentieren, dass einige bösartige Benutzer besonders Effizient sein. Als Begründung führen Sie an, dass daran interessiert sind ihre Aktionen besonders häufig, schnell und in großen Umfang ausgeführt werden. Neben der Länge und der maximalen Anzahl an wiederholenden Buchstaben im Benutzernamen

verwendeten sie ebenfalls die folgenden Meta-Features zur Erfassung der Effizienz eines Benutzers.

Die folgenden Meta-Feature untersuchen die Anordnung der Buchstaben in den Eingaben auf der Tastatur. Betrachtet wird der Benutzername, die E-Mail-Adresse, der Echtname und die Homepage. Als Tastaturmodell wurde zum einen das deutsche QWERTZ-Modell gewählt, da fast die hälfte der Nicht-Spammer aus Deutschland kommen. Als Alternatives Layout wurde DOVRAK gewählt. Um zu entscheiden welcher Finger, welche Taste gedrückt hat, wird angenommen der Benutzer tippt nach dem Modell des 10 Fingersystem. Abbildung 5.16 zeigt die Zuordnung der einzelnen Finger zu den jeweiligen Tasten im 10 Fingersystem.



(a)emailProp3RowDvorak



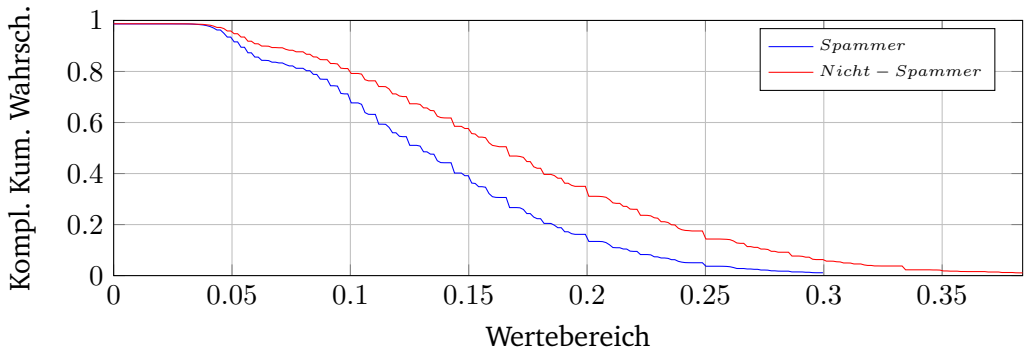
(b)realnameProp4RowQwertz

Abb. 5.17.: Empirische Verteilungsfunktionen im Meta-Feature $propRow$

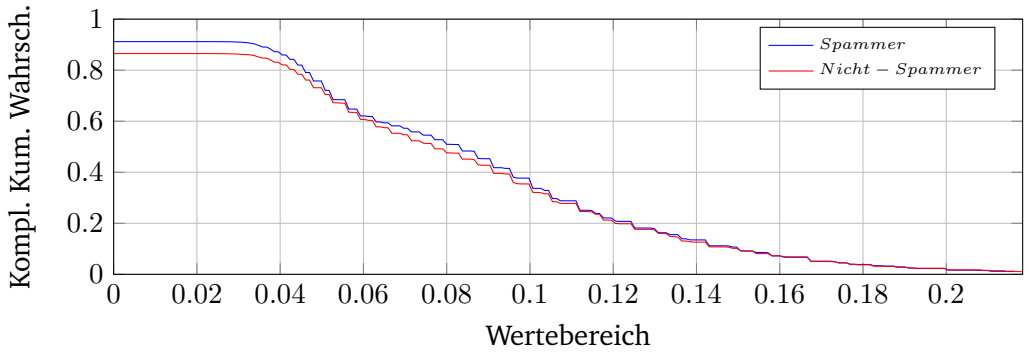
Das Meta-Feature $propRow$ beschreibt den Anteil an Tasten in einer Reihe. Als Reihen wird die Zahlenreihe, sowie die drei darunter liegenden Reihen betrachtet. Kontroll-Tasten und Funktionstasten werden nicht betrachtet. Aus den vier Reihen kombiniert mit den vier Eingabefeldern und zwei Tastaturmodellen ergeben sich aus diesem Meta-Feature 32 Features. Abbildung 5.17 zeigt, dass es z.B. bei dem Qwertz Layout bei der E-Mail-Adresse in der dritten Reihe deutliche Unterschiede zu erkennen gibt, während bei dem Echtnamen in der vierten Reihe kaum Unterschiede

zu erkennen sind. Insgesamt sind die Unterschiede zwischen den Verteilungen der Werte zwischen Spammern und Nicht-Spammern eher gering.

Das Meta-Feature $propFinger$ beschreibt den Anteil an Tasten auf dem selbem Finger. Es wird der Kleine-, Ring-, Mittel- und Zeigefinger der linken und rechten Hand unterschieden. Der Daumen wird nicht betrachtet, da dieser lediglich das Leerzeichen im 10-Finger-Schreiben bedient. Somit ergeben sich aus diesem Meta-Feature 64 Features. Abbildung 5.18 zeigt, dass Nicht-Spammer dazu tendieren häufiger Tasten den linken Mittelfinger zu benutzen, während es beim rechten Mittelfinger, kaum Unterschiede gibt. Insgesamt gibt es vereinzelnd Features, die deutliche Unterschiede zeigen. Bei der Mehrheit der Features im Meta-Feature $propFinger$ sind die Unterschiede in den Verteilungsfunktionen zwischen Spammern und Nicht-Spammern gering.



(a)emailPropMiddleFingerLeftQwertz

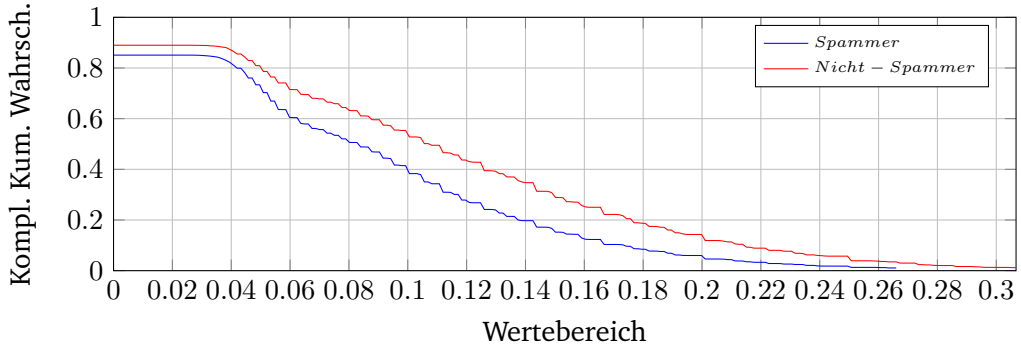


(b)emailPropMiddleFingerRightQwertz

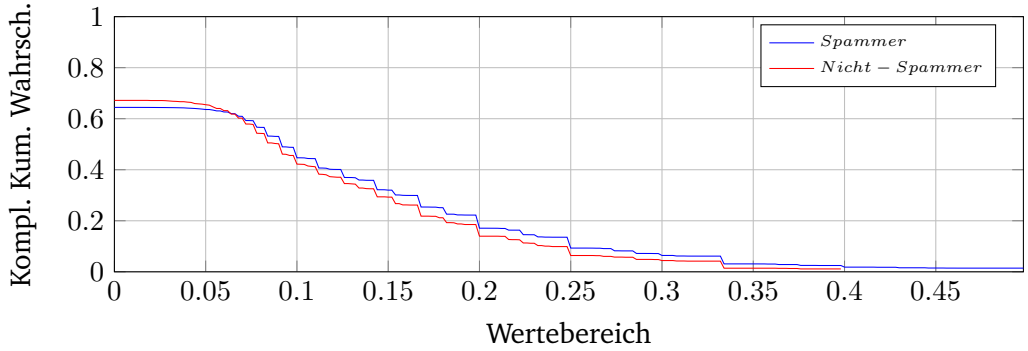
Abb. 5.18.: Empirische Verteilungsfunktionen des Meta-Feature $propFinger$

Das Meta-Feature $propSameFingerAsPrev$ beschreibt den Anteil an Tasten, die mit dem selben Finger, wie die vorherige Taste, getippt wurden. Es wird nach den gleichen Fingern, wie bereits bei $propFinger$ unterschieden. Bei der E-Mail-Adresse (5.19 a), der Homepage und dem Benutzernamen sind Unterschiede zu

erkennen. Angewendet auf den Echtnamen (5.19 b) ergeben sich keine signifikanten Unterschiede zwischen Spammern und Nicht-Spammern.



(a)emailPropSameFingerAsPrevQwertz

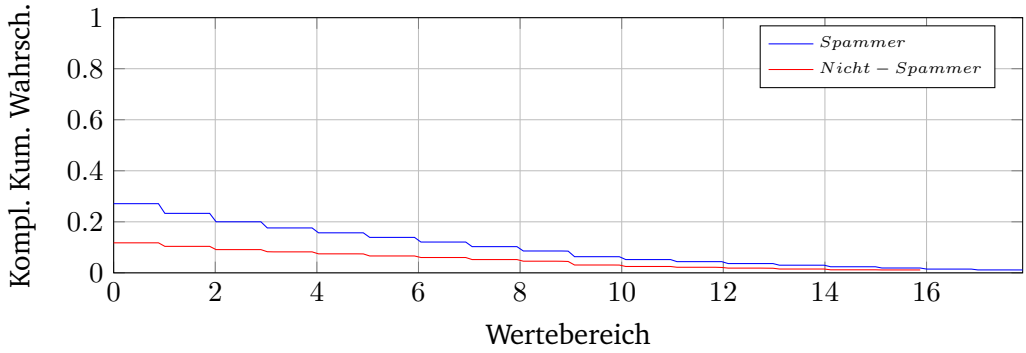


(b)realnamePropSameFingerAsPrevQwertz

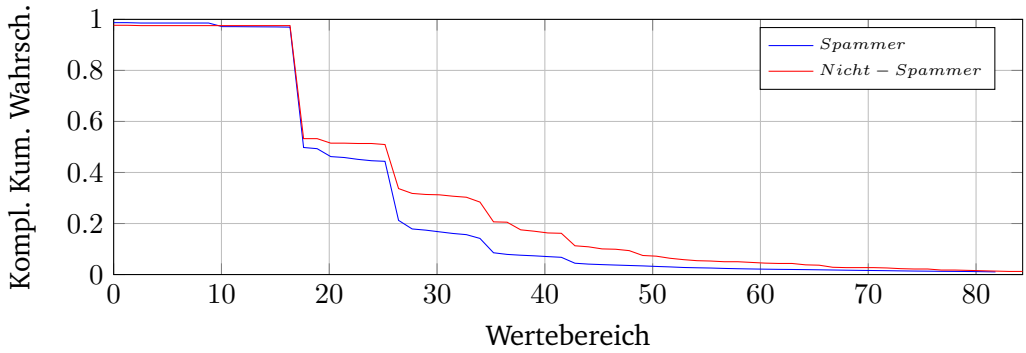
Abb. 5.19.: Empirische Verteilungsfunktionen des Meta-Feature $propSameFingerAsPrev$

Das Meta-Feature $propSameHandAsPrev$ beschreibt den Anteil von Tasten, die mit derselben Hand, mit der auch die vorherige Taste getippt wurde, gedrückt wurde. Es wird nach der linken und rechten Hand unterschieden. In den Verteilungsfunktionen der Features zeigen sich bei der E-Mail-Adresse und der Homepage geringe Unterschiede zwischen Spammern und Nicht-Spammern. Bei dem Benutzernamen und dem Echtnamen sind nur minimalste Unterschiede vorhanden.

Das Meta-Feature $distanceOnKeyboard$ beschreibt die approximierte Distanz, die sich ergeben würde, wenn man mit einem Finger Taste für Taste die Eingabe eintippen würde. Es wird angenommen, dass Zwei nebeneinanderliegende Tasten 1cm auseinander liegen. Die empirischen Verteilungsfunktionen zeigen, dass Nicht-Spammer zu höheren Distanzen bei der E-Mail-Adresse, der Homepage (Abbildung 5.20 a) und dem Echtnamen tendieren. Spammer tendieren zu größeren Distanzen bei dem Benutzernamen (Abbildung 5.20 b).



(a)nameDistanceOnKeyboardQwertz



(b)homepageDistanceOnKeyboardDvorak

Abb. 5.20.: Empirische Verteilungsfunktionen des Meta-Feature $distanceOnKeyboard$

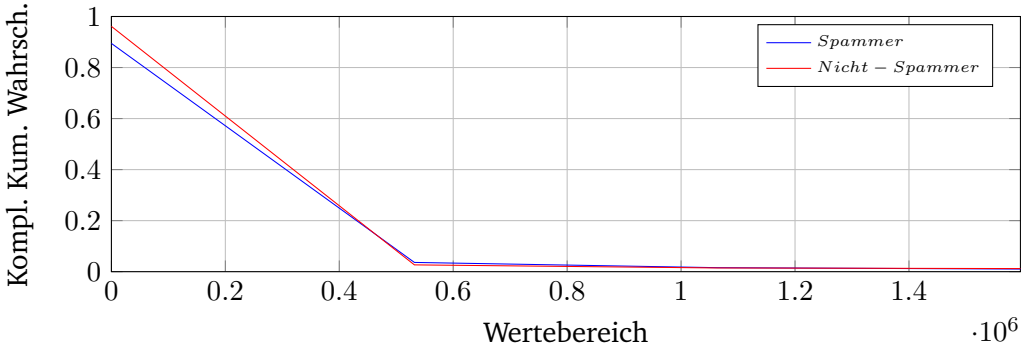
Zusammenfassend lassen die empirischen Verteilungsfunktion auf einige erfolgreiche Features hindeuten. Viele der Features haben jedoch auch sehr ähnliche Verteilungsfunktionen und werden vermutlich keine große Verbesserung der Klassifikatoonsgüte bringen. Maßnahmen zum Filtern der relevanten Features sind daher für diese Kategorie von Features zwingend empfohlen. Die hierfür eingesetzten Methoden werden im nachfolgenden Kapitel erläutert.

5.5 Auf Log-Daten basierende Features

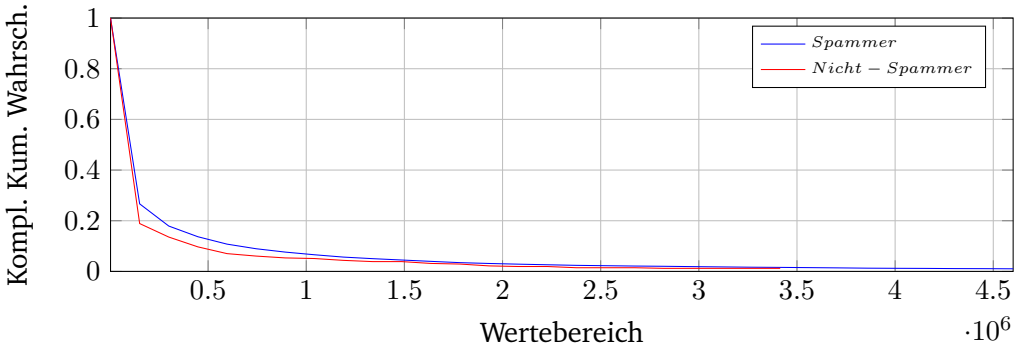
Die in diesem Abschnitt beschriebenen Features benutzen Informationen benutzen, die aus den Apache-Log-Daten rekonstruiert wurden (siehe Kaptiel 4.3). Nach entsprechender Änderung des Bibsonomy Quellcodes können diese auch auf die normalen Benutzerdaten von Bibsonomy angewendet werden.

Im vorherigen Abschnitt wurden die Auswertung des Tippverhaltens aufgrund der Effizient von bösartigen Benutzern implementiert. Dieses lässt sich ebenso auf die Dauer der Registrierung und Aktivierung übertragen. Das Feature $registrationDuration$

gibt die Zeit zwischen Anfordern der Webseite zur Registrierung und klicken auf Registrieren an. Hat der Benutzer, aufgrund von Fehlern in der Eingabe, mehrmals auf Registrieren klicken müssen, so wird das letzte Klicken auf Registrieren gewertet. Das Featue $activationDuration$ beschreibt die Zeitdauer zwischen erfolgreicher Registrierung und erfolgreicher Aktivierung. Abbildung 5.21 zeigt die empirischen Verteilungsfunktionen der Feature $registrationDuration$ & $activationDuration$. In beiden Feature gibt es keinen signifikanten Unterschied zwischen den Verteilungen der Spammer und Nicht-Spammer.



(a)registrationDuration



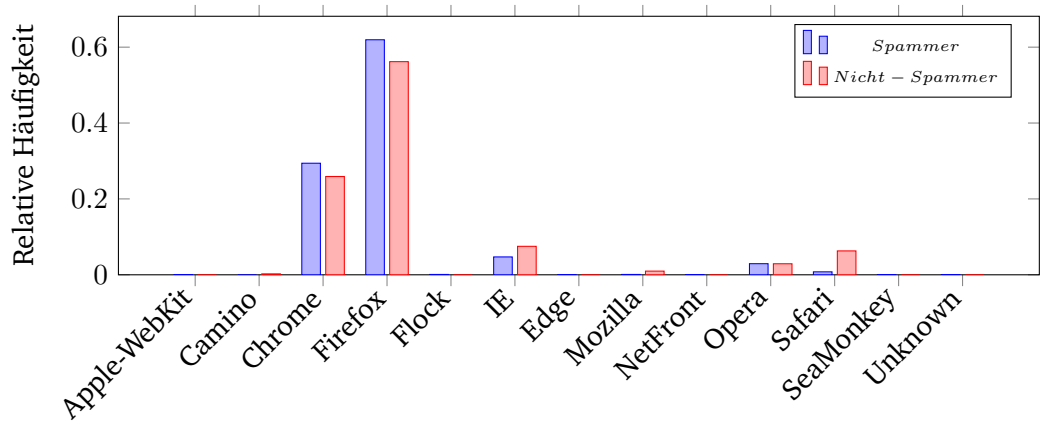
(b)activationDuration

Abb. 5.21.: Empirische Verteilungsfunktionen der Feature $registrationDuration$ & $activationDuration$

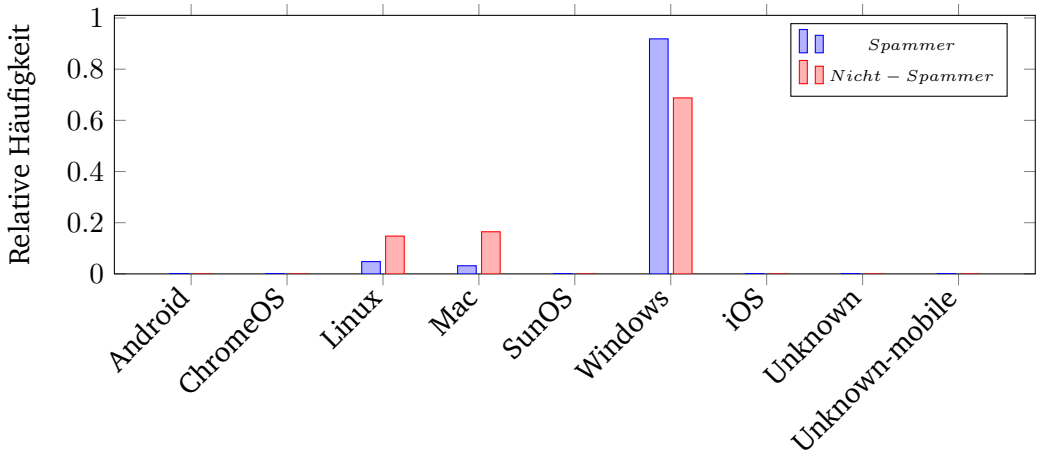
In Kapitel 5.2 wurde bereits ein automatisierter Registrierungsprozess angedeutet. In diesem Registrierungsprozess kann bzw. muss, denn andernfalls wäre dieses zu auffällig, ein User-Agent angegeben werden. In diesem kann ein Browser und ein Betriebsystem angegeben sein. Diese können aufgrund der technischen Entwicklung nicht immer gleich bleiben. Es könnte also passieren, dass solche Prozesse veralten und von den Angaben von den anderen Benutzern abweichen. Das Feature $browser$ beschreibt den im User-Agent angegebenen Browser. Das Feature os beschreibt das im User-Agent angegebene Betriebssystem. Abbildung 5.22 a zeigt, dass die meisten Spammer auf Chrome und Firefox als Browser zurückgreifen. Die Nicht-Spamer

hingegen nutzen ein breiteres Sprektrum an verschiedenen Browsern. Ähnlich verhält es sich beim Betriebssystem, wie Abbildung 5.22 b zeigt. Hier nutzen 91% der Spammer Windows, während nur 68% der Nicht-Spammer Windows benutzen. Die zwei weiteren relevanten Betriebssysteme bei den Nicht-Spammern sind Linux und Mac OS X.

Zusätzlich gibt das Feature $refererFromBibsonomy$ an, ob der angegebene Referer von Bibsonomy stammt. Der Referer gibt die Seite an, von der aus die Webseite, hier die Registrierung, aufgerufen wurde. Hierbei sei anzumerken, dass Webserver üblicherweise nicht überprüfen, ob der Referer korrekt ist. In diesem Feature sind jedoch keine relevanten Unterschiede zuerkennen. Bei 98% der Spammer und 99% der Nicht-Spammer war der Referer von Bibsonomy.



(a)browser



(b)os

Abb. 5.22.: Werte-Verteilungen der Feature $browser$, os

Zusammenfassend zeigen sich in dieser Kategorie lediglich Unterschiede zwischen Spammern und Nicht-Spammern im verwendeten Browser und Betriebssystem.

Alle anderen Features zeigen keinen signifikanten Unterschiede in den empirischen Verteilungsfunktionen der Spammer und Nicht-Spammer.

5.6 Interaktions-basierende Features

Das Maus- und Tippverhalten wurde bereits erfolgreich zu Identifikation von Benutzern eingesetzt [18]. Zur Beschreibung der Mausbewegung wurden vor allem Geschwindigkeit, Beschleunigung, zurückgelegte Distanz und Zeit, in der die die Maus bewegt wurde, benutzt. Feher et. al. [12] schlagen vor zudem die gemessenen Verhaltensweise nach Richtungen zu Unterscheiden. Daher werden alle nachfolgend genannten Features jeweils für 8 verschiedene Richtungen betrachtet. Die Richtung ergibt sich aus dem Winkel des Vektors zwischen zwei Mauspunkten zum Vektor $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

Die Geschwindigkeit, Beschleunigung und benötigte Zeit hängt stark von der verwendeten Hardware in der Maus ab. Ein Lichtsensor mit hoher Empfindlichkeit überwindet bei gleicher Distanz auf dem Mauspad mehr Pixel als eine Maus mit einem weniger empfindlichen Sensor. Um diesen entgegen zu wirken, werden die Werte je Richtung mit dem maximalen Wert über alle Richtungen normiert. Bewegen sich nun zwei Mäuse mit der exakt gleichen Geschwindigkeit so wird nicht mehr der Unterschied in der Hardware erfasst, da ein relatives Verhalten pro Richtung erfasst wird und nicht der absolute Wert.

Das Meta-Feature $velocity$ erfasst die Geschwindigkeit der Maus. Dabei wird zwischen je zwei Datenpunkt vom Event-Typ $mousemove$ die Geschwindigkeit als Distanz zwischen den Zwei Datenpunkten in Pixeln bezogen auf die Zeitdifferenz zwischen den zwei Events erfasst. Um den Fehler, bedingt durch Pausen, zu verringern werden Geschwindigkeiten zwischen zwei Punkten, deren zeitliche Differenz eine Sekunde oder höher ist, nicht betrachtet. Untersuchungen in den Neurowissenschaften haben gezeigt, dass Menschen im Durchschnitt 400ms Reaktionszeit benötigen, um ihren Arm zu bewegen, nachdem sie mit einer Nadel gestochen wurden [4]. Nach 400ms müsste demnach sich die Maus, seit der letzten Bewegung, wieder bewegt haben, wenn der Benutzer in einer Bewegung abrupt abbrechen und sofort wieder die Hand bewegen würde. Als Schutzabstand wird diese Reaktionszeit auf eine Sekunde erhöht. Mit einer Sekunde sollte man demnach sicher sein, dass die Bewegung unterbrochen wurde. Es wird je Richtung die minimale, maximale und durchschnittliche Geschwindigkeit betrachtet, welche Anschließend mit den maximalen Wert über alle Richtungen normiert werden. Insgesamt ergeben sich aus diesem Meta-Feature 24 konkrete Features.

Das Meta-Feature $acceleration$ erfasst die Beschleunigung der Maus. Die Beschleunigung wird dabei über alle Events vom Typ $mousemove$ ermittelt. Ebenfalls werden Beschleunigungen zwischen zwei Punkten, deren zeitliche Differenz eine Sekunde oder höher ist, ignoriert. Die Beschleunigung wird wie folgt berechnet. Am Punkt P_i liegt die Geschwindigkeit V_i, welche durch die Punkte P_{i-1} und P_i berechnet wurden, an. Die Beschleunigung ist die zeitliche Änderung der Geschwindigkeit zwischen den Punkten P_i und P_{i+1}. Es wird je Richtung die minimale, maximale und durchschnittliche Beschleunigung betrachtet, welche Anschließend mit den maximalen Wert über alle Richtungen normiert werden. Insgesamt ergeben sich aus diesem Meta-Feature 24 konkrete Features.

Das Meta-Feature $movedDistance$ erfasst die bewegten Pixel der Maus. Diese wird je Richtung zwischen je Zwei Punkten vom Event-Typ $mousemove$ aufsummiert und anschließend mit dem maximalen Wert über alle Richtungen normiert. Insgesamt ergeben sich aus diesem Meta-Feature 8 konkrete Features.

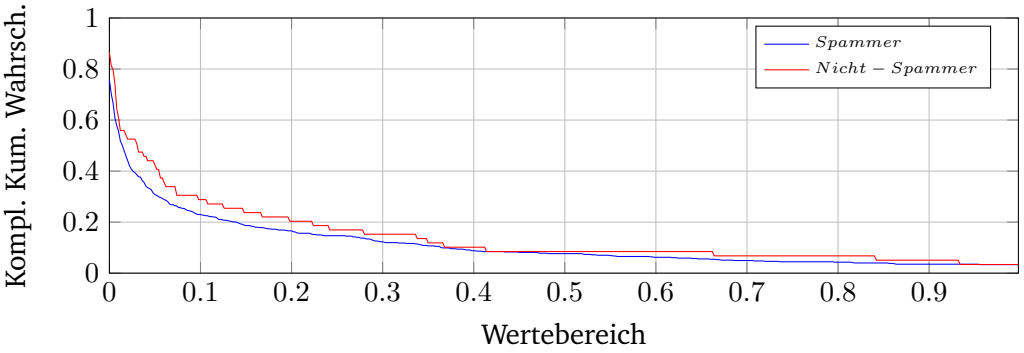
Feher et. al. [12] benutzten zudem die vergangene Zeit und aufgebrachte Distanz zwischen dem drücken und dem loslassen der Maus. Zudem wurde zwischen Rechts-, Links-, Doppelklick und Drag and Drop unterschieden. Diese Unterscheidung ist im Falle von Bibsonomy nicht notwendig. Bei der Registrierung gibt es weder Felder die einen Linksklick erfordern, noch Felder die einen Doppelklick erfordern. Daher wurde diese zusammengefasst und es wird nur betrachtet, ob geklickt wurde. Gleiches gilt für Drag and Drop. Dieses Funktion ist bei der Registrierung nicht vorgesehen.

Das Meta-Feature $pauseNClick$ erfasst die Zeit zwischen dem Ende der Bewegung und dem drücken der Maustaste. Hierzu werden alle Events der Typen $mousemove$ und $mousedown$ betrachtet. Ist ein Event am Punkt P_i vom Typ $mousemove$ und am Punkt P_{i+1} vom Typ $mousedown$, so ist die Zeit zwischen diesen beiden Punkten die Zeit zwischen dem Ende der Bewegung und dem drücken der Maustaste. Es wird die minimale, maximale und durchschnittliche Zeit erfasst.

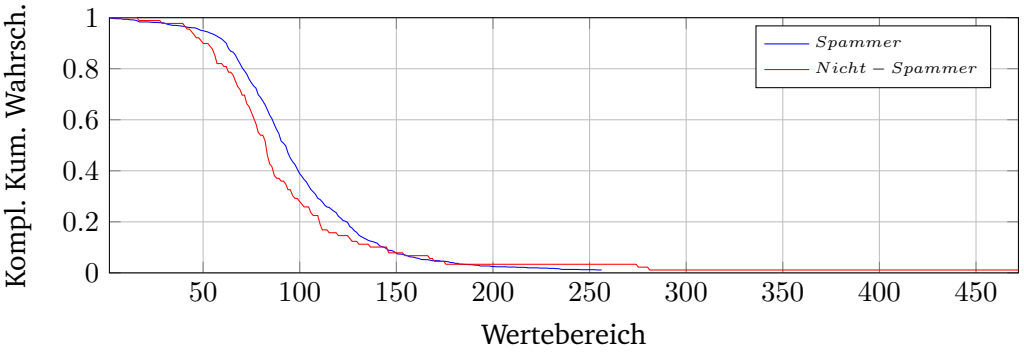
Das Meta-Feature $clickTime$ erfasst die Zeit, die zum drücken einer Taste benötigt wird. Hierzu werden alle Events der Typen $mouseup$ und $mousedown$ betrachtet. Ist ein Event am Punkt P_i vom Typ $mousedown$ und am Punkt P_{i+1} vom Typ $mouseup$, so ist die Zeit zwischen diesen beiden Punkten die Zeit, die zum Drücken der Maustaste benötigt wird. Es wird die minimale, maximale und durchschnittliche Zeit erfasst.

Die Analyse des Tippverhaltens wird mithilfe von Zeitabständen zwischen zwei aufeinander folgenden Tastendrücken vollzogen [24]. Das Meta-Feature $dwellTime$ erfasst die Zeit zwischen dem Drücken einer Taste und dem Loslassen einer Taste. Hierzu werden alle Events der Typen $keydown$ und $keyup$ betrachtet. Folgt ein

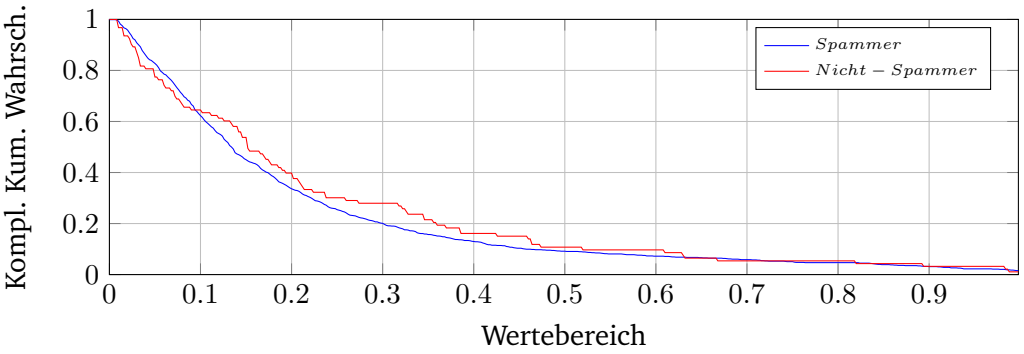
Event vom Typ $keyup$ auf ein Event vom Typ $keydown$ und haben beide den gleiche Key-Code, so ist die Zeit zwischen diesen beiden Events die Dwell Time. Es wird die minimale, maximale und durchschnittliche Zeit erfasst. Das Meta-Feature $flightTime$ erfasst die Zeit zwischen dem Loslassen einer Taste und dem Dücken der nächsten Taste.



(a)meanAcceleration7



(b)meanDwellTime



(c)movedDistance0

Abb. 5.23.: Empirische Verteilungsfunktionen der Feature $meanAcceleration7$, $meanDwellTime$ & $movedDistance0$

Hierzu werden alle Events der Typen $keydown$ und $keyup$ betrachtet. Folgt ein Event vom Typ $keydown$ auf ein Event vom Typ $keyup$, so ist die Zeit zwischen diesen beiden Events die Flight Time. Um den Fehler, bedingt durch Pausen, zu verringern wird die Flight Time zwischen zwei Punkten, deren zeitliche Differenz eine Sekunde oder höher ist, nicht betrachtet. Es wird die minimale, maximale und durchschnittliche Zeit erfasst.

Die Auswertung der empirischen Verteilungsfunktionen der Features in dieser Kategorie ergab, dass sich bei keinem Feature ein signifikanter Unterschied in den Verteilungen ergeben hat. Es ist also anzunehmen, dass es keinen wesentlichen Unterschied zwischen Spammern und Nicht-Spammern gibt, wie die Daten bei der Registrierung eingegeben werden. Abbildung 5.23 zeigt Beispielhaft die empirischen Verteilungsfunktionen der Feature $meanAcceleration7$, $meanDwellTime$ & $movedDistance0$.

6 Methodik

Dieses Kapitel beschreibt zum einen Methoden, sowie dessen Konfiguration, die zum ausführen und evaluieren der Klassifikation notwendig sind. Zum anderen werden Methoden und dessen Parameter beschrieben, die die Güte der Klassifikation erhöhen können.

6.1 Klassifikatoren

Die Auswahl der Klassifikatoren orientiert sich an dessen erfolgreichen Einsatz in bisherigen wissenschaftlichen Veröffentlichungen. Zur Klassifikation der Spammer werden Logistic-Regression, Support Vector Machine (SVM), J48, und Naive Bayes benutzt. Alle wurden bereits erfolgreich bei der Detektion von Spam eingesetzt [10, 31, 3, 21, 23]. Für SVM wird ein linearer Kernel benutzt, da dieser besonders zeiteffizient ist. Für Logistic-Regression, J48 und Naive Bayes wurden die Weka-Implementierungen implementierungen benutzt. Für SVM wurde die LibLinear-Implementierung benutzt.

6.2 Feature Selection

Für viele Probleme in der Anwendung sind die relevanten Features, um diese zu beschreiben nicht bekannt. Es ist daher notwendig, diese über eine Vielzahl von Features zu beschreiben. Unglücklicherweise verschlechtern irrelevante Features die Klassifikation, im Sinne von benötigter Zeit und Genauigkeit der Klassifikation [19]. Um diese irrelevanten Features herauszufiltern werden Feature Selection Techniken benutzt. Als Feature Selection beschreibt man das Problem, aus einer Menge von Feature jene Teilmenge auszuwählen, die idealerweise notwendig und ausreichend ist, um die zu klassifizierende Beobachtung zu beschreiben[19].

Yang et. al [30] untersuchten mehrere Feature Selection Techniken in Kontext der Kategorisierung von Texten. In ihren Untersuchungen hat sich die Auswahl der besten Features nach dem Informationsgewinn und der Chi-Square-Statistik bewehrt. Im Bereich der Erkennung von E-Mail-Spam haben Sharma et. al [26] ebenfalls erfolgreich die Chi-Square-Statistik zur Feature Selection benutzen können. Sie argumentieren zudem, dass Feature Selection nach der Chi-Square-Statistik

sehr Zeiteffizient im Vergleich zu anderen Feature-Selection-Techniken sei. Toolan et. al [27] testeten den Informationsgewinn als Feature Selection Technik, mit 40 klassische Features aus der Literatur zur Spam- und Phishing-Erkennung. Sie evaluierten die besten, mittleren und schlechtesten Features, gewertet nach ihrem Informationsgewinn. Die Features mit dem bestem Informationsgewinn brachten auch die besten Klassifikations-Ergebnisse.

Aufgrund ihres Erfolgs in der Literatur werden für diese Arbeit als Kriterien zur bestimmung der besten Feature der Informationsgehalt und die Chi-Square-Statistik benutzt. Der Informationsgewinn gibt an, um wieviel sich die Entropy der Beobachtung reduzieren würde, würde man die Werte des Features kennen [30]. Die Chi-Square-Statistik gibt den Grad an statistischer Abhängigkeit zwischen dem Feature und der Klasse an [30].

6.3 Class Imbalance

In dem vorhandenen Datensatz sind etwa 50 mal mehr Spammer vorhanden, als Nicht-Spammer. Diese ungleiche Verteilung der Klassen wird auch als Class Imbalance Problem bezeichnet. Class Imbalance kann durchaus einen negativen Effekt auf die Güte von Klassifikatoren haben [17].

Eine Möglichkeit dem ungleichen Verhältnis von Spammern zu Nicht-Spammern entgegenzuwirken sind Kosten-Sensitive Klassifikatoren [21]. Diese gewichten die einzelnen Klassen so, dass Fehler auf einer Klasse schwerer wiegen kann, als auf der anderen Klasse. In Weka gibt es die Möglichkeit den Meta-Klassifikator CostSensitiveClassifier zusammen mit einem anderen Klassifikator zu benutzen. Diesem kann eine Kosten-Matrix übergeben werden. In dieser ist angegeben welche Kosten eine Entsprechende Klassifikation verursacht. Es werden die Auswirkungen von den einfachen bis zu zehnfachen Kosten für Falsch-Negative Ergebnisse gegenüber Falsch-Positiven Ergebnissen betrachtet.

6.4 Evaluation

Zur Evaluation der Klassifikation hat sich eine 10-fache stratifizierte Kreuzvalidierung bewehrt [20]. Hierzu wurde zunächst die von Weka bereitgestellte Implementierung benutzt. Algorithmus 4 zeigt diese in Pseudocode. Die Reihenfolge der Daten im Datensatz wird zunächst zufällig angeordnet, um eventuelle Anhäufungen von ähnlichen Daten zu vermeiden. Daraufhin werden die Daten stratifiziert. So wird sichergestellt, dass die Verteilung der Klassen in jedem Abschnitt möglichst die gleiche Verteilung der Klassen besitzt, wie der gesamte Datensatz. Daraufhin folgt

die eigentliche Kreuzvalidierung. Der Datensatz wird in k, möglichst gleich große Abschnitte unterteilt. Mit diesen Abschnitten wird der Klassifikator k mal trainiert und getestet. Im i-ten Durchlauf wird dabei der i-te Abschnitt zum testen und die restlichen Abschnitte zum trainieren benutzt.

Algorithmus 4 k-fache stratifizierte Kreuzvalidierung

```
1: procedure CROSSVALIDATEMODEL(classifier, data, k)
2:     data ← randomize(data)
3:     data ← stratify(data)
4:     for fold from 1 to k do
5:         train ← trainCV(data, fold)
6:         test ← testCV(data, fold)
7:         classifier.build(train)
8:         test(classifier, test)
9:     end for
10: end procedure
```

Werden die Populations-basierten Features jedoch mit allen Instanzen als Population erzeugt und evaluiert, erzeugt dieses einen Fehler in den Ergebnissen. Der Grund hierfür soll an dem nachfolgenden Beispiel erläutert werden.

Tab. 6.1.: Beispiel-Daten: Feature-Daten wurden aus allen Daten erzeugt

	Spammer	IP-Adresse	$SpamIP$
1	1	1.1.1.1	0,5
2	0	2.2.2.2	0
3	0	1.1.1.1	0,5
4	1	1.1.1.1	0
5	1	1.1.1.1	0,5

Tab. 6.2.: Beispiel-Daten: Feature-Daten wurden aus einem Teil der Daten erzeugt

	Spammer	IP-Adresse	$SpamIP$
1	1	1.1.1.1	1
2	0	2.2.2.2	0
3	0	1.1.1.1	1
4	1	1.1.1.1	1
5	?	1.1.1.1	1

Tabelle 6.1 zeigt fünf verschiedene Benutzer mit der Angabe, ob es sich um einen Spammer handelt, dessen IP-Adresse und das Feature SpamIP (Siehe Kapitel 5.3). Es gibt zwei Spammer mit der IP-Adresse 1.1.1.1 und einen Spammer mit der

IP-Adresse 3.3.3.3. Diese haben also entsprechend einen Wert bei $SpamIP$ von 0,5 (Ein anderer Spammer von zwei anderen Spammern) bzw. 0. Bei den Nicht-Spammern hat ein Benutzer die IP-Adresse 1.1.1.1 und ein Anderer die IP-Adresse 2.2.2.2. Entsprechend haben diese also einen $SpamIP$ Wert von 0,5 bzw. 0.

Tabelle 6.2 zeigt, dass der Klassifikator in Wirklichkeit die Features diese Werte jedoch nicht annehmen würden. Angenommen die Benutzung haben sich entsprechend der Reihenfolge, in der diese in der Tabelle stehen registriert und nach jeder Registrierung wurde der Klassifikator neu trainiert. Zum Zeitpunkt der Registrierung von Benutzer 5 kann in den Daten nicht enthalten sein, dass dieser als Spammer die IP-Adresse 1.1.1.1 hat. Entsprechend wird aus der IP-Adresse 1.1.1.1 ein $spamIP$ -Wert von 1. Analoges gilt bei der Kreuzvalidierung. Ist der Benutzer nicht in der Trainingsmenge, darf er auch bei dem Feature $spamIP$ nicht berücksichtigt werden.

Um dieses Problem zu vermeiden, müssen in jedem Durchlauf der Kreuzvalidierung die Features mit den Trainingsdaten initialisiert werden und die Trainings- und Testdaten so erzeugt werden, dass nur die Trainingsdaten als Gesamtheit der Population angenommen wird. Algorithmus 5 zeigt die modifizierte k-fache stratifizierte Kreuzvalidierung. Neben dem Klassifikator, den Daten und der Anzahl der Durchläufe wird der Kreuzvalidierung zusätzlich eine Zuordnung übergeben, die angibt welche Daten von welchen Benutzern erzeugt wurde. Da bei der Randomisierung und der Stratifizierung die Reihenfolge der Daten verändert werden, müssen diese dahingehend geändert werden, dass bei der Vertauschung der Daten, die Zuordnung entsprechend mit geändert wird. In jedem Durchlauf müssen bei Populations-basierte Features die Gesamtheit der Benutzer neu gesetzt und dessen Werte entsprechend neu gesetzt werden. Mithilfe der Zuordnung von Daten zu Benutzer, kann eine Liste von Benutzern erzeugt werden, sodass an der i-ten Position der Liste der User zu finden ist, der den i-ten Feature-Vektor der Trainings- bzw. der Testdaten erzeugt hat. Anschließend können die Werte entsprechend aktualisiert werden.

Algorithmus 5 Modifizierte k-fache stratifizierte Kreuzvalidierung

```
1: procedure CROSSVALIDATEMODEL(classifier, data, k, users, userAss)
2:     data, userAss ← randomize(data, userAss)
3:     data, userAss ← stratify(data, userAss)
4:     for fold from 1 to k do
5:         train ← trainCV(data, fold)
6:         users ← generateTrainUsers(fold, users, userAss)
7:         for each feature f do
8:             if fneedsallusers then
9:                 f.setUsers(users)
10:                for i from 0 to users.size − 1 do
11:                    f.apply(users.get(i), train.get(i))
12:                end for
13:            end if
14:        end for
15:        test ← testCV(data, fold)
16:        users ← generateTestUsers(fold, users, userAss)
17:        for each feature f do
18:            if fneedsallusers then
19:                for i from 0 to users.size − 1 do
20:                    f.apply(users.get(i), test.get(i))
21:                end for
22:            end if
23:        end for
24:        classifier.build(train)
25:        test(classifier, test)
26:    end for
27: end procedure
```

7 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Experimente dargestellt. Zunächst werden als Ausgangssituation die Ergebnisse von allen Features, mit den in Kaptitel 6.1 vorgestellten Klassifikatoren, dargestellt. Ausgehend von diesen Ergebnissen werden die Fragen geklärt, welchen Einfluss das Class-Imbalance-Problem auf den Datensatz hat und wie sich die Feature-Selection auf die Güte der Klassifikation auswirkt. Anschließend werden die Ergebnisse abschließend zusammengefasst und eine Empfehlung für die Konfiguration der Anbindung an Bibsonomy gegeben.

Tabelle 7.1 zeigt die Evaluationsergebnisse für Features der Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basierte Feature, welche mit dem Datensatz aus Kapitel 4.1 erzeugt wurden. Es ist der Klassifikator, die Klasse (S = Spammer, NS = Nicht-Spammer, G = Gewichtetes Mittel), die Precision (Prec.), der Recall (Rec.) das F1-Maß (F1) die Area Under ROC (AUC), die Anzahl der richtig und falsch klassifizierten Instanzen, sowie die für die 10-fache Kreuzvalidierung benötigte Zeit dargestellt.

Tab. 7.1.: Evaluations-Ergebnisse für alle Feature der Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basierte Feature

Klassifikator	Klasse	Prec.	Rec.	F1	AUC	Richtig	Falsch	t [s]
Logistic Regression	NS	0,65	0,34	0,45	0,91	1242	2372	1305
Logistic Regression	S	0,99	1,00	0,99	0,91	179721	655	1305
Logistic Regression	G	0,98	0,98	0,98	0,91	180963	3027	1305
Naive Bayes	NS	0,07	0,62	0,13	0,78	2255	1359	115
Naive Bayes	S	0,99	0,84	0,91	0,78	151029	29347	115
Naive Bayes	G	0,97	0,83	0,89	0,78	153284	30706	115
J48	NS	0,76	0,22	0,34	0,66	800	2814	2928
J48	S	0,98	1,00	0,99	0,66	180121	255	2928
J48	G	0,98	0,98	0,98	0,66	180921	3069	2928
SVM	NS	0,82	0,19	0,31	0,60	691	2923	293
SVM	S	0,98	1,00	0,99	0,60	180222	154	293
SVM	G	0,98	0,98	0,98	0,60	180913	3077	293

Logistic Regression liefert mit 0,91 den besten AUC-Wert, sowie mit 3027 falsch klassifizierten Benutzern, den geringsten Klassifikationsfehler insgesamt. In der

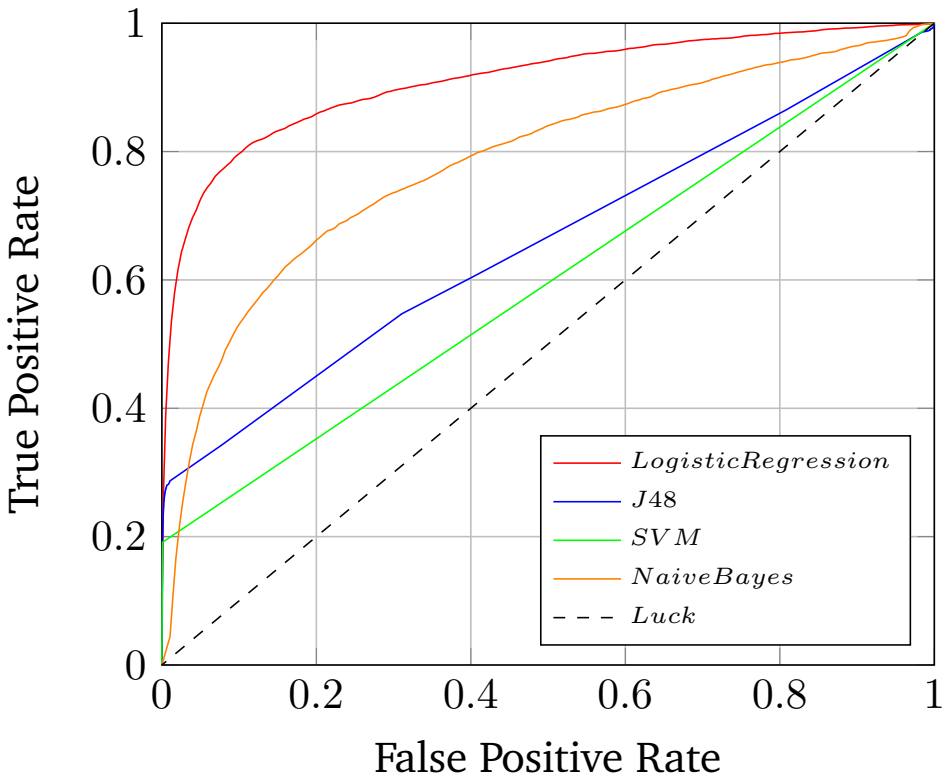


Abb. 7.1.: ROC für alle Feature der Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basierte Feature

Klasse der Nicht-Spammer macht der Naive Bayes Klassifikator, mit 1359 falsch klassifizierten Benutzern, den geringsten Klassifikationsfehler. Zudem ist der Naive Bayes Klassifikator der schnellste der verwendeten Klassifikatoren. J48 und SVM liefern mit einem AUC-Wert von 0,66 bzw. 0,60 eher schlechte Ergebnisse. Um zu ermitteln bei welchen Falsch-Positiv-Raten, welche Richtig-Positiv-Raten zu erzielen sind, sind in Abbildung 7.1 die ROC-Kurven der einzelnen Klassifikatoren abgebildet. Für Logistic Regression gilt dabei: Bei 1% Fehlklassifikation der in der Klasse der Nicht-Spammer, werden ca. 50% der Spammer richtig klassifiziert werden. Bei 5% Fehlklassifikation in der Klasse der Nicht-Spammer werden ca. 72% der Spammer richtig klassifiziert. Naive Bayes klassifiziert bei 5% Fehlklassifikation in der Klasse der Nicht-Spammer, 38% der Spammer richtig. Bei J48 sind es 27% und bei SVM 24%. Da die Fehlklassifikation von Nicht-Spammer gering gehalten werden sollte, ist ein Wert zwischen 1% und 5% Fehlklassifikation empfehlenswert. Bei den Sprach-, Umwelt-, Tastatur- und Populations-basierte Features ist somit Logistic Regression als Klassifikator zu empfehlen.

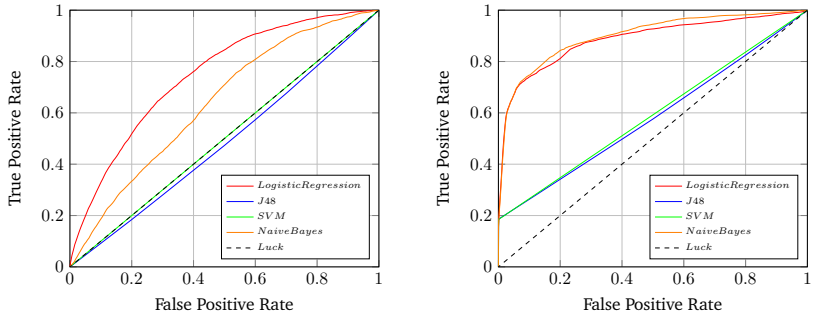
Tabelle 7.2 zeigt den Einfluss der einzelnen Feature-Kategorien. Es wird je Kategorie der Klassifikator, dargestellt, der den besten AUC-Wert in der Kategorie erreichte. Den besten AUC-Wert erzielte dabei Naive Bayes mit einem AUC-Wert von 0,90 in der Kategorie Umwelt-basierte Feature. Darauf folgen die Tastatur- und Sprach-

basierten Features mit einem AUC-Wert von 0,81 bzw. 0,75, jeweils mit Logistic Regression. Lediglich die Populations-basierten Features unterscheiden sich mit einem AUC-Wert von 0,50 nicht vom Zufall. Die wahrscheinlichste Erklärung hierfür ist, dass im Meta-Feature $count$ sehr viele Nicht-Spammer einen Wert von Null haben. Lediglich einige Spammer haben höhere Werte. Diese Feature können also einen Spammer definieren, jedoch nicht einen Nicht-Spammer. Auch die Feature im Meta-Feature $ratio$ sind auch keine allzu starken Feature, wie die Auswertung der Feature Selektion, später in diesem Kaptitel, zeigt. Insgesamt hat dieses anscheinend zur Folge, dass die Klassifikatoren Spammer und Nicht-Spammer nicht unterscheiden können. So klassifizieren diese alle Benutzer als Spammer, da so der geringere Fehler entsteht.

Tab. 7.2.: Evaluations-Ergebnisse nach Kategorien

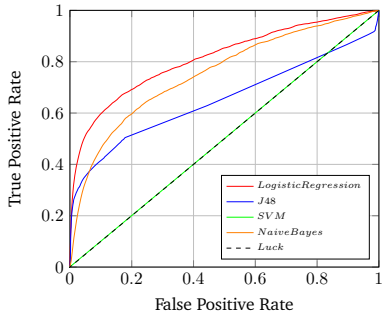
Kategorie	Bester Klassifikator.	AUC
Umwelt	Naive Bayes	0,90
Tastatur	Logistic Regression	0,81
Sprache	Logistic Regression	0,75
Population	Logistic Regression, J48, SVM	0,50

Abb. 7.2.: ROC je Kategorie

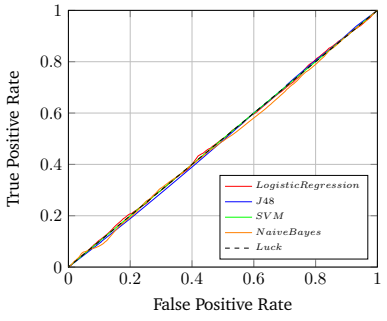


(a)ROC Sprach-basiert

(b)ROC Umwelt-basiert



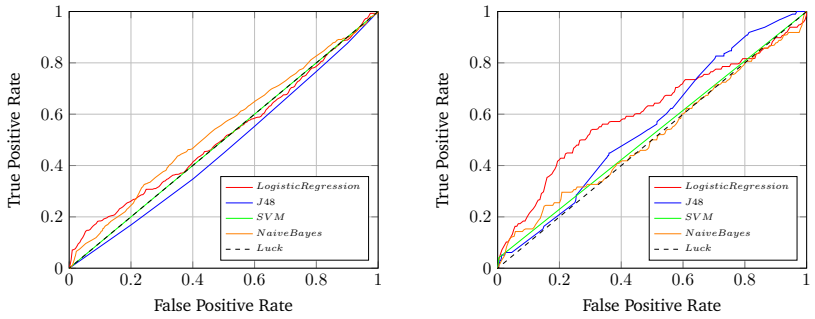
(c)ROC Tastatur-basiert



(d)ROC Populations-basiert

Abbildung 7.2 zeigt die ROC-Kurven der einzelnen Kategorien. Es ist zu sehen, dass die Umwelt-basierten Features fast die Ergebnisse aller Features, der vier

Abb. 7.4.: ROC der Kategorien auf Log-Dateien basierte und Interaktions-basierte Feature



(a)ROC Auf Log-Dateien basiert **(b)**ROC Interaktions-basiert

Kategorien zusammen, erreichen. Der Naive Bayes Klassifikator klassifiziert, bei 1% Fehlklassifikation in der Klasse der Nicht-Spammer, ca. 35% der Spammer richtig . Bei bei 5% Fehlklassifikation in der Klasse der Nicht-Spammer werden ca. 67% der Spammer richtig klassifiziert.

Die auf Log-Daten- und Interaktions-basierten Features müssen mit jeweils einzelnen Datensätzen erzeugt werden. Die auf Log-Daten-basierten Features wurden mit jenen Benutzern aus dem vorherigen verwendeten Datensatz, denen sich eine Registrierung aus den Apache-Log-Daten zuordnen ließ, evaluiert. Die Interaktions-basierten wurden mit Benutzern evaluiert, deren Registrierung, nach Implementierung eines Loggers, aufgezeichnet wurde. Beide Kategorien verbessern die Klassifikationsgüte nicht wesentlich. Abbildung 7.4 zeigt, dass bei beiden Kategorien die ROC-Kurven sich nicht wesentlich vom Zufall unterscheiden. Bei den auf Log-Dateien basierenden Features ist dieses zum einen auf die geringe Anzahl an Features zurückzuführen. Zum anderen unterscheiden sich die empirischen Verteilungsfunktionen zwischen Spammern und Nicht-Spammern nur signifikant bei den Features $browser$ und os . Bei den Interaktions-basierenden Features unterschieden sich die empirischen Verteilungsfunktionen bei keinem Feature zwischen Spammer und Nicht-Spammer. Somit ist auch keine große Klassifikationsgüte von dieser Kategorie zu erwarten gewesen.

Um den Einfluss von Class Imbalance zu untersuchen wurden die zu untersuchenden Klassifikatoren in Kosten-Sensitive Klassifikatoren eingehüllt. Mit diesen wurde untersucht, wie sich die Klassifikationsgüte verändert, wenn die Kosten für die Fehlklassifikation eines Nicht-Spammers vom ein- bis zum zehfachen der Kosten, der Fehlklassifikation eines Spammers, variiert. Tabelle 7.3 zeigt für jedes dieser Vielfachen den besten Klassifikator, sowie dessen AUC-Wert.

Tab. 7.3.: Evaluations-Ergebnisse der Kosten-Sensitiven-Klassifikatoren

FP Kosten	FN Kosten	Bester Klassifikator.	AUC
1	1	Logistic Regression	0,912
1	2	Logistic Regression	0,912
1	3	Logistic Regression	0,909
1	4	Logistic Regression	0,914
1	5	Logistic Regression	0,913
1	6	Logistic Regression	0,910
1	7	Logistic Regression	0,905
1	8	Logistic Regression	0,906
1	9	Logistic Regression	0,905
1	10	Logistic Regression	0,903

Zwischen den verschiedenen Kosten für eine Falsch-Negative Klassifikation sind nur marginale Unterschiede vorhanden. Den besten AUC mit 0,914 liefert der Kosten-Sensitive-Klassifikator, der Falsch-Negative Klassifikationen wird vier mal höher bewertet, als eine Falsch-Positive-Klassifikation. Dieser klassifiziert, bei 1% Fehlklassifikation in der Klasse der Nicht-Spammer, ca. 50% der Spammer richtig . Bei bei 5% Fehlklassifikation in der Klasse der Nicht-Spammer werden ca. 73% der Spammer richtig klassifiziert.

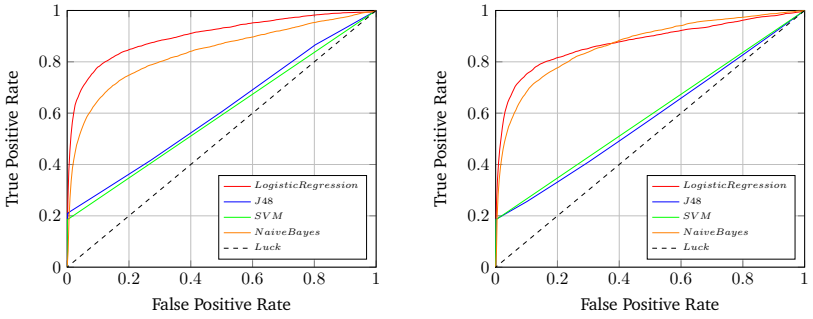
Ein wichtiger Aspekt ist die Dauer der Klassifikation, damit Bibsonomy die Aktivierung abschließen kann, ohne dass der Benutzer diese mitbekommt. Daher ist Feature Selection notwendig. Es wurde Klassifikationsgüte für die 10, 25, 50 und 75 besten Features, gemessen am Informationsgewinn (IG) und der Chi-Sqaure-Statistik (ChSq), ermittelt. Dabei wurde der Datensatz in Kapitel 4.1 und den Featue der Kategorien Features der Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basierte Feature benutzt. Tabelle 7.4 zeigt jeweils den Klassifikator mit dem besten AUC-Wert, dem Kriterium der Feature Selection, die Anzahl der Features, die richtig und falsch klassifizierten Benutzer, sowie die für die 10-fache Kreuzvalidierung benötigte Zeit. Das Ergebnis ist, dass bereits die 25 besten Features fast die gleiche Güte wie alle Features gesamt erreichen.

Tab. 7.4.: Evaluations-Ergebnisse der Feature Selection

Klassifikator	Kriterium	#Feature	AUC	Richtig	Falsch	t [s]
Logistic Regression	IG	10	0,88	180868	3122	382
Logistic Regression	IG	25	0,90	180904	3086	717
Logistic Regression	IG	50	0,90	180852	3138	984
Logistic Regression	IG	75	0,90	180892	3098	1244
Logistic Regression	ChiSq	10	0,88	180828	3162	342
Logistic Regression	ChiSq	25	0,90	180890	3100	593
Logistic Regression	ChiSq	50	0,90	180879	3111	950
Logistic Regression	ChiSq	75	0,90	180846	3144	1228

Abbildung 7.6 zeigt die ROC-Kurven für die 25 besten Features nach Informationsgewinn und Chi-Square-Statistik. Mit dem Informationsgewinn als Kriterium klassifiziert Logistic Regression, bei 1% Fehlklassifikation in der Klasse der Nicht-Spammer, ca. 48 % der Spammer richtig klassifiziert. Bei einer Fehlklassifikation von 5% in der Klasse der Nicht-Spammer, werden 70% der Spammer richtig klassifiziert. Das sind jeweils nur 2 Prozent-Punkte weniger als bei allen Features, bei nur ca. 54% der benötigten Zeit. Mit der Chi-Square-Statistik als Kriterium klassifiziert Logistic Regression, bei 1% Fehlklassifikation in der Klasse der Nicht-Spammer, ca. 42 % der Spammer richtig klassifiziert. Bei einer Fehlklassifikation von 5% in der Klasse der Nicht-Spammer, werden 67% der Spammer richtig klassifiziert. Das sind 8 bzw. 5 Prozent-Punkte weniger als bei allen Features, bei nur ca. 45% der benötigten Zeit.

Abb. 7.6.: ROC der 25 besten Feature nach IG und ChSq



(a)ROC IG **(b)**ROC ChSq

Insgesamt ist Logistic Regression als Klassifikator zu empfehlen, da dieser sich in allen Untersuchungen als einer der besten Klassifikatoren herausgestellt hat. Es hat sich gezeigt, dass es einen Vorteil von einem Prozentpunkt bringt, wenn ein Kosten-Sensitiver Klassifikator eingesetzt wird. Bei diesem ist ein FN zu FP Kosten Verhältnis von eins zu vier zu empfehlen. Bei der Auswahl der Feature sollten die 25 besten

Features, gemessen am Informationsgewinn, eingesetzt werden. Diese bietet einen guten Kompromiss zwischen benötigter Zeit und Klassifikationsgüte. So müssen ca. 2 Prozentpunkte der richtigen Klassifikation von Spammern, für fast 50% Zeitersparniss, eingebüßt werden. Die 25 besten Features nach dem Informationsgewinn über alle Feature-Kategorien sind in Tabelle 7.5 abgebildet. Da der Datensatz aus Kapitel 4.1 die besten Grunddaten liefert, wurden die Untersuchungen zur Class Imbalance und Feature Selection auf diesen und damit auf die Feature der Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basiert angewendet. Dessen 25 beste Feature, nach dem Informationsgewinn, sind in Abbildung 7.6 abgebildet.

Tab. 7.5.: Die 25 besten Features absteigend sortiert nach ihrem Informationsgewinn

Feature	IG
country	0,039396
uniMail	0,018293
emailDistanceOnKeyboardQwertz	0,015035
emailDistanceOnKeyboardDvorak	0,014066
spamIP	0,012551
os	0,011629
emailPropMiddleFingerLeftDvorak	0,010904
emailPropRingFingerLeftDvorak	0,010617
browser	0,005391
emailInfoSupr	0,004724
firstnameRatio	0,004680
emailPropMiddleFingerLeftQwertz	0,004582
emailPropLittleFingerLeftQwertz	0,003601
emailPropRingFingerRightQwertz	0,003595
nameProp1RowQwertz	0,003584
namePropDigit	0,003584
emailProp1RowDvorak	0,003570
nameProp1RowDvorak	0,003560
emailProp1RowQwertz	0,003279
emailPropDigit	0,003275
emailNumDigit	0,003270
nameNumDigit	0,003216
nameDigit	0,003179
emailProp3RowDvorak	0,003166
emailDigit	0,002976

Tab. 7.6.: Die 25 besten Features aus den Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basiert absteigend sortiert nach ihrem Informationsgewinn

Feature	IG
country	0,039396
uniMail	0,018293
emailDistanceOnKeyboardQwertz	0,015035
emailDistanceOnKeyboardDvorak	0,014066
spamIP	0,012551
emailPropMiddleFingerLeftDvorak	0,010904
emailPropRingFingerLeftDvorak	0,010617
emailInfoSupr	0,004724
firstnameRatio	0,004680
emailPropMiddleFingerLeftQwertz	0,004582
emailPropLittleFingerLeftQwertz	0,003601
emailPropRingFingerRightQwertz	0,003595
nameProp1RowQwertz	0,003584
namePropDigit	0,003584
emailProp1RowDvorak	0,003570
nameProp1RowDvorak	0,003560
emailProp1RowQwertz	0,003279
emailPropDigit	0,003275
emailNumDigit	0,003270
nameNumDigit	0,003216
nameDigit	0,003179
emailProp3RowDvorak	0,003166
emailDigit	0,002976
nameLength	0,002678
emailPropMiddleFingerRightDvorak	0,002581

8 Anbindung an Bibsonomy

Dieses Kapitel beschreibt die technische Umsetzung der Klassifikation von Benutzern zur Aktivierung in Bibsonomy. Die Klassifizierung muss zum einen in kurzer Zeit stattfinden. Eine zu lange Klassifizierungsdauer, würde die Benutzbarkeit von Bibsonomy einschränken, da die Ladezeit der Rückmeldung, dass die Aktivierung erfolgreich war, entsprechend verzögert wird [9]. Zum anderen muss die Klassifizierung zu jeder Zeit und mit beliebig vielen Benutzer gleichzeitig arbeiten können. Die Details der Umsetzung werden im folgenden erläutert:

Abbildung 8.1 zeig eine Übersicht der für die Anbindung der Klassifizierung an Bibsonomy implementierten Klassen als UML-Klassendiagramm. Diese Besteht im wesentlichen aus drei Bereichen. Der Implementierung der Feature-Struktur ($AbstractFeature$), welche bereits in Kapitel 5 beschrieben wurde, die Verwaltung und Erzeugung der Featuredaten aus den Benutzerdaten ($SimpleSpammerDetectionManager$) und der Realisierung der Schnittstelle zu Bibsonomy ($RegistrationSpammerDetectionServlet$).

Die Verwaltung der Features und der Erzeugung von Instance- und Instances-Objekten, die mit Weka klassifiziert werden können, übernimmt der $SimpleSpammerDetectionManager$. Eine besondere Anforderung an diese Klasse ist, dass die Klassifizierung von Benutzern zum einen parallel stattfinden kann. Zum anderen muss gewährleistet werden, dass zur Laufzeit der Klassifikator neu erzeugt und trainiert werden kann. Die Parallelität ist sehr einfach umzusetzen. Da bei einem trainiertem Klassifikator, dieser durch die Klassifikation eines Benutzers nicht verändert wird, kann auf diesen ohne Anpassung parallel zugegriffen werden. Um die Neubildung des Klassifikators Fehlerfrei stattfinden lassen zu können, wird ein Zähler von aktuell klassifizierenden Prozessen gepflegt. Zudem hat der $SimpleSpammerDetectionManager$ eine Referenz auf einen Wartungs-Thread. Ist dieser nicht auf $Null$ gesetzt, warten alle anderen neuen Prozesse mit der Klassifizierung von Benutzern, bis dieser beendet ist. Soll nun der Klassifikator neu trainiert werden, so geschieht dies zunächst auf einer parallelen Classifier-Instanz. Ist diese traininert wird ein Wartungs-Thread gestartet. Dieser wartet solange bis der Zähler von aktuell klassifizierenden Prozessen auf Null ist. Anschließend wird die neue Classifier-Instanz gesetzt und die Referenz des Wartungs-Thread auf $Null$ gesetzt.

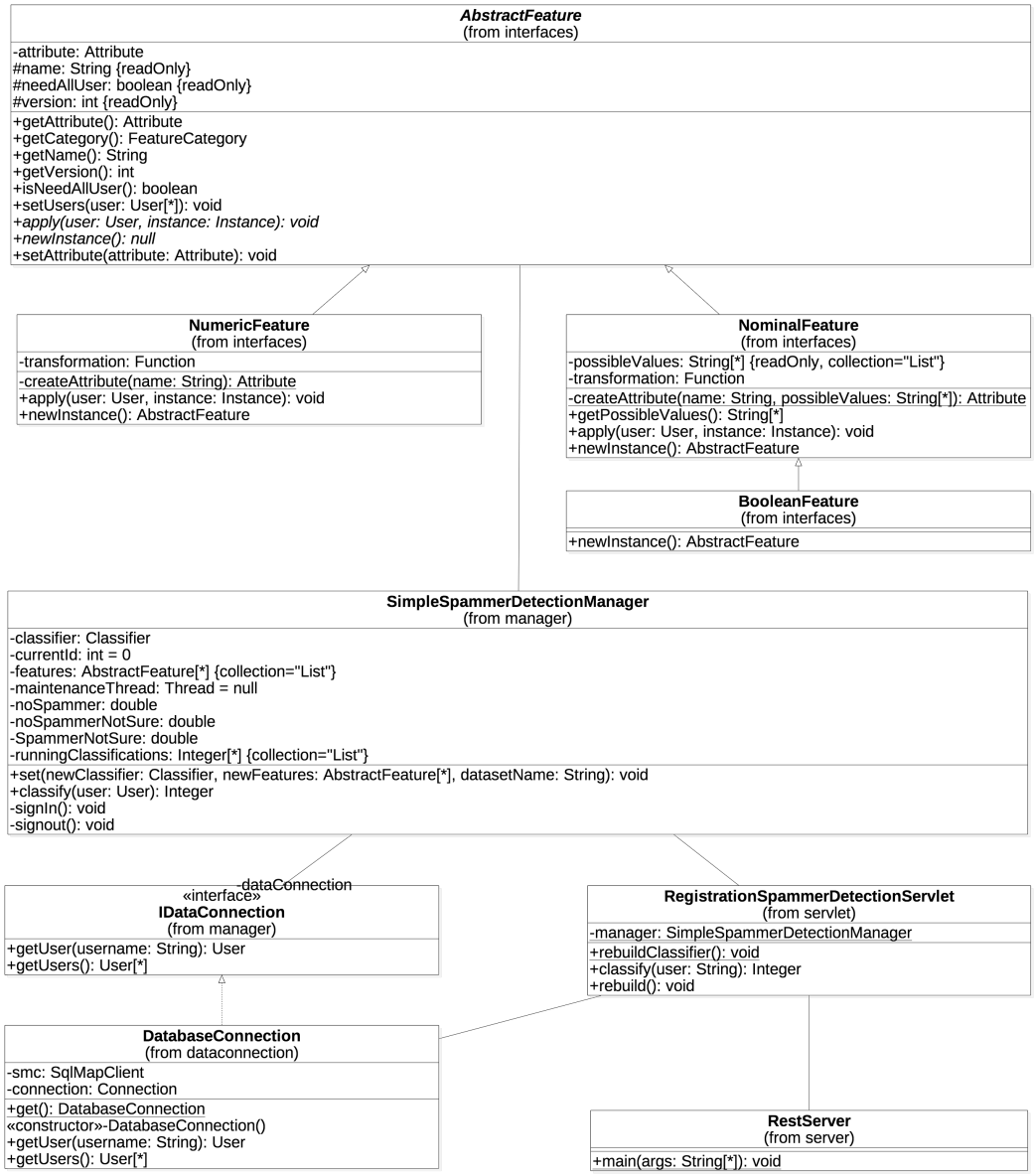


Abb. 8.1.: Klassendiagramm Anbindung an Bibsonomy

Die Schnittstelle zu Bibsonomy wurde mittels eines embedded Jetty-Rest-Server realisiert. Dieses Vorgehen hat den Vorteil, dass die Klassifikation entkoppelt von Bibsonomy läuft. Dieses ermöglicht es den Service neu zu starten, um z.B. neue Features zu integrieren, ohne dass Bibsonomy neu gestartet werden muss. Zudem können beide Anwendungen auf verschiedenen Systemen laufen, sodass die CPU-Last beider Systeme einfacher zu Händeln sind. Zusätzlich beeinflusst ein Absturz des Klassifikationssstems, bis auf die Möglichkeit Spammer bei der Registrierung zu klassifizieren, nicht Bibsonomy. Die Klasse $RegistrationSpammerDetectionServlet$ realisiert die konkrette Schnittstelle mittels der GET Methoden $classify$ und $rebuild$. Wird classify mit einem Benutzer als Parameter aufgerufen, so wird die Methode $classify$ einer statischen Instanz des $SimpleSpammerDetectionManager$ aufgerufen und das Ergebnis zurückgesendet. Wird $rebuild$ aufgerufen, so wird $rebuild$ des $SimpleSpammerDetectionManager$ mit den in der Konfiguration hinterlegten Standardwerte für Klassifikator und dessen Optionen, sowie die in der Konfiguration festgesetzten Features, aufgerufen Der Activation-Controller in Bibsonomy nach erfolgreicher Aktivierung den Rest-Server mittels HTTP-GET auf und aktualisiert den Spam-Status des Benutzers entsprechend. Ist der Rest-Server nicht erreichbar oder kann entsprechend festgelegtes Zeitlimit nicht einhalten werden, wird der Spamstatus auf den Standard-Wert gesetzt und als zu Klassifizieren markiert.

Die Auswahl des Klassifikators, der Feature, der Feature Selektion und Class Imbalance Optionen kann über eine Konfigurationsdatei bestimmt werden. Zusätzlich kann in dieser eine Zeit-Regel definiert werden, die bestimmt zu welchen Zeitpunkten der eingesetzte Klassifikator mit den aktuellen Benutzern aus Bibsonomy neu trainiert werden soll.

8.1 Benchmark

Um die Zeiteffizienz der Implementierung zu evaluieren, wird die Zeit von 1000 Anfragen gemessen. Dazu verwendet der Test und die Implementierung den Datensatz aus Kapitel 4.1. Aus diesen Daten wird 1000 mal je ein Benutzer zufällig ausgewählt und per HTTP-Get $classifiy$ des REST-Servers aufgerufen. Hat der Rest-Server geantwortet wird der nächste Benutzer ausgewählt und klassifiziert. Zudem wurde die Zeit von 1000 $rebuild$ aufrufen gemessen. Als Klassifikator wird jeweils Logistic Regression mit den 25 besten Features, gemessen am Informationsgewinn, gewählt. In beiden Versuchen ist der Datensatz als Liste von Benutzern im Arbeitsspeicher. Es wird also die reine Bearbeitungszeit der Klassifikation betrachtet. In der späteren Benutzung kommen zu den ermittelten Zeiten noch die Dauser für die Übertragung zwischen Mysql-Server, sowie die Verarbeitung der Anfragen auf dem Mysql-Server hinzu. Dieses ist im Rahmen dieser Arbeit nicht zu testen.

Der für die Messungen benutzte Computer hat die folgenden Spezifikationen:

- CPU: Intel i5 3570k, Grundtakt: 3,4 Ghz, Max. Taktfrequenz 3,8 Ghz
- Arbeitspeicher: 8 GB DDR-3 1600
- Betriebssytem: Mac OS 10.11
- Java: Oracle JDK 1.8.0_77

Tab. 8.1.: Benchmark REST-Server

Anfrage	Dauer Minimal	Dauer Mittel	Dauer Maximal
$classify$	5ms	18ms	457ms
$rebuild$	3m 52s	4m 3s	4m 17s

Tabelle 8.1 zeigt, dass im mittel $classify$ im Mittel ca. 18 Milisekunden und $rebuild$ ca. 4 Minuten benötigt. Es ist zu beachten, dass diese Zeiten je nach verwendeter Hardware, und dem Netzwerk zwischen der Datenquelle, dem Rest-Server und dem Bibsonomy-Webserver variieren können. Es ist daher zu Empfehlen bei der Inbetriebnahme der Implementierung die Zeiten erneut gemessen werden, um das Timeout des HTTP-Requests entsprechend einstellen zu können.

9 Fazit

Diese Arbeit hat gezeigt, dass es möglich ist Benutzer in einem kollaborativen Verschlagwortungssystem, wie Bibsonomy, bereits zur Registrierung bzw. zur Aktivierung zu klassifizieren. Dabei wurden 270 Features entwickelt bzw. recherchiert, implementiert und untersucht. Diese Feature entstehen zum einen aus Informationen die der Benutzer an Bibsonomy übermittelt, wie die Eingaben bei der Registrierung, der Art und Weise, wie diese Eingaben bei der Registrierung gemacht werden und dem Umfeld des Benutzers, wie der IP-Adresse. Während die Eingaben selbst und vor allem das Umfeld zu sehr effektiven Features führen, hat sich gezeigt, dass bei der Art der Eingabe zwischen Spammern und Nicht-Spammern keine großen Unterschiede vorliegen. Die größe des Datensatz könnte jedoch zu einem gewissen Fehler geführt haben. Daher ist es lohnenswert die diesbezüglichen Evaluationen, zu einem späteren Zeitpunkt mit einem größeren Datensatz zu wiederholen. Insgesamt konnte im besten Fall ein AUC-Wert von 0,91 erreicht werden. Im Vergleich dazu hat das bereits existierende Framework im besten Fall einen AUC von 0,93 erreicht [21]. Im unterschied zum bisherigen Ansatz, können die erkannten Spammer jedoch von Anfang an, davon abgehalten werden Spam zu verbreiten. Der bereits existierende Ansatz muss die Spammer zunächst Spam in Bibsonomy verbreiten lassen, bevor dieses jene als Spammer klassifizieren kann. Zudem muss der in dieser Arbeit entwickelte Ansatz nicht als Nachfolger oder Konkurrent des bisherigen Ansatzes gesehen werden. Vielmehr ergänzt dieser den bisherigen Ansatz. Darüber hinaus wurde ein REST-Server zur Klassifizierung von Benutzern zur Aktivierung implementiert, sodass die gewonnen Erkenntnisse im laufenden Betrieb von Bibsonomy zeitnah eingesetzt werden können.

In zukünftigen Arbeiten kann zum einen versucht werden die Klassifikationsgüte weiter zu verbessern. Zafarani et. al. [31] haben weitere interessante Ansätze vorgeschlagen, die jedoch Aufgrund der begrenzten Zeit nicht weiter untersucht werden konnten. So haben diese z.B. versucht den Benutzern demographische Daten, wie dem alter des Benutzers, zuzuordnen, um so Unterschiede zwischen normalen und bösartigen Benutzern zu erkennen.

Neben der Erhöhung der Klassifikationsgüte kann auch versucht werden die Kosten von Spam für Spammer zu erhöhen. Ein bereits existierender Ansatz ist, dass in Bibsonomy Spammer vor dem Hinzufügen von Links oder Veröffentlichungen ein ReCaptcha ausfüllen müssen. Dieses könnte erweitert und Bibsonomy spezifisch

ausgebaut werden. Anstatt dass der Spammer, wie im klassischen ReCaptcha Bilder zuordnen oder Wörter eingeben muss, könnten z.B. häufige Tags Veröffentlichungen zugeordnet werden. Somit könnten gleichzeitig die getätigten Tags überprüft werden.

Ein weiterer in diese Richtung gehende Ansatz ist ein Tutorial für Bibsonomy. Den Benutzern wird Interaktiv die Funktionen von Bibsonomy beigebracht. So könnte dieser z.B. neue Veröffentlichungen hinzufügen oder diese mit Tags versehen. Der unterschied zwischen als Spammer klassifizierten und normalen Benutzern besteht darin, dass Spammer das Tutorial machen müssen. Alle anderen könnten dieses auch überspringen. Damit wäre Bibsonomy in dreierlei Weise geholfen. Erstens lernen neue Benutzer, wie Bibsonomy zu bedienen ist. Zweitens wird er Inhalt von Bibsonomy aufgewertet. Und drittens ergibt sich so die Möglichkeit mehr Informationen, über die Benutzer zu sammeln, um eine eventuelle Falsch-Klassifikation noch korrigieren zu können.

A Anhang

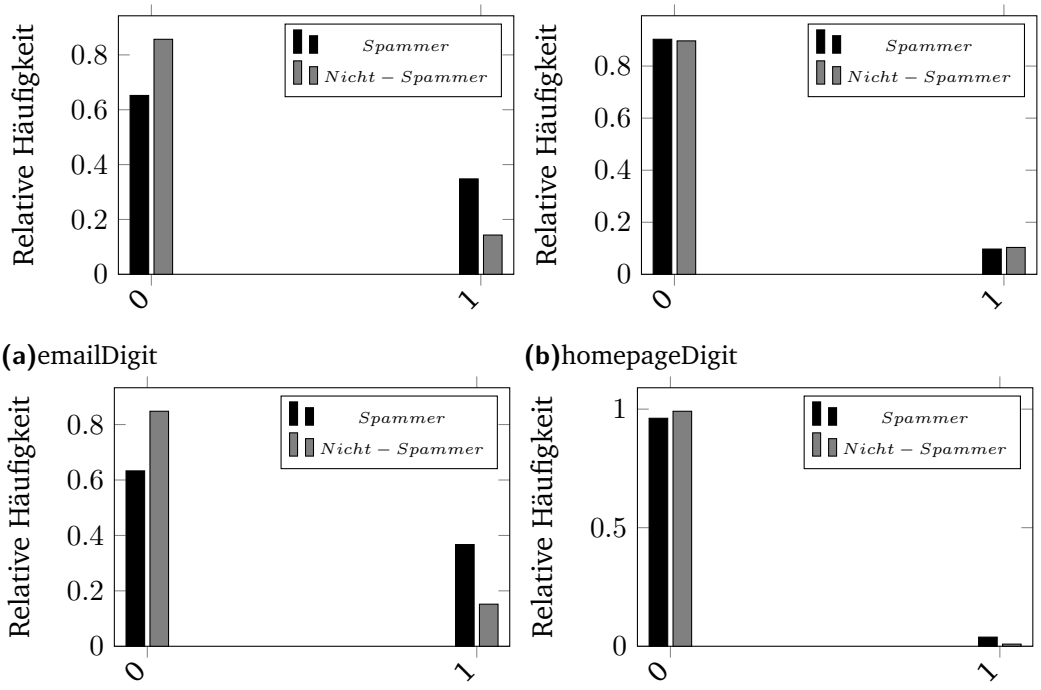
A.1 Sprach-basierte Features

Tab. A.1.: Sprach-basierte Features 1 / 2

Feature Name	IG	Chi-Square
$emailDigit$	0,002976	658,16
$emailEntropy$	0,001431	434,34
$emailInfoSupr$	0,004724	2224,53
$emailLength$	0,002171	977,95
$emailNumDigit$	0,003270	686,70
$emailNumMaxTimesLetterRep$	0,000302	103,34
$emailNumStartingDigits$	0,000000	0,00
$emailNumUniqueAlphLetters$	0,000627	201,61
$emailPropDigit$	0,003275	872,81
$emailPropStartingDigits$	0,000000	0,00
$homepageDigit$	0,000000	0,07
$homepageEntropy$	0,000000	0,00
$homepageInfoSupr$	0,000037	9,63
$homepageLength$	0,000000	0,00
$homepageNumDigit$	0,000000	0,00
$homepageNumMaxTimesLetterRep$	0,000000	0,00
$homepageNumStartingDigits$	0,000000	0,00
$homepageNumUniqueAlphLetters$	0,000010	2,56
$homepagePopStartingDigits$	0,000000	0,00
$homepagePropDigit$	0,000000	0,00
$nameDigit$	0,003179	709,42
$nameEntropy$	0,002274	725,51
$nameEqualMail$	0,000545	130,64
$nameEqualMailEqualRealnname$	0,000151	33,64

Tab. A.2.: Sprach-basierte Features 2 / 2

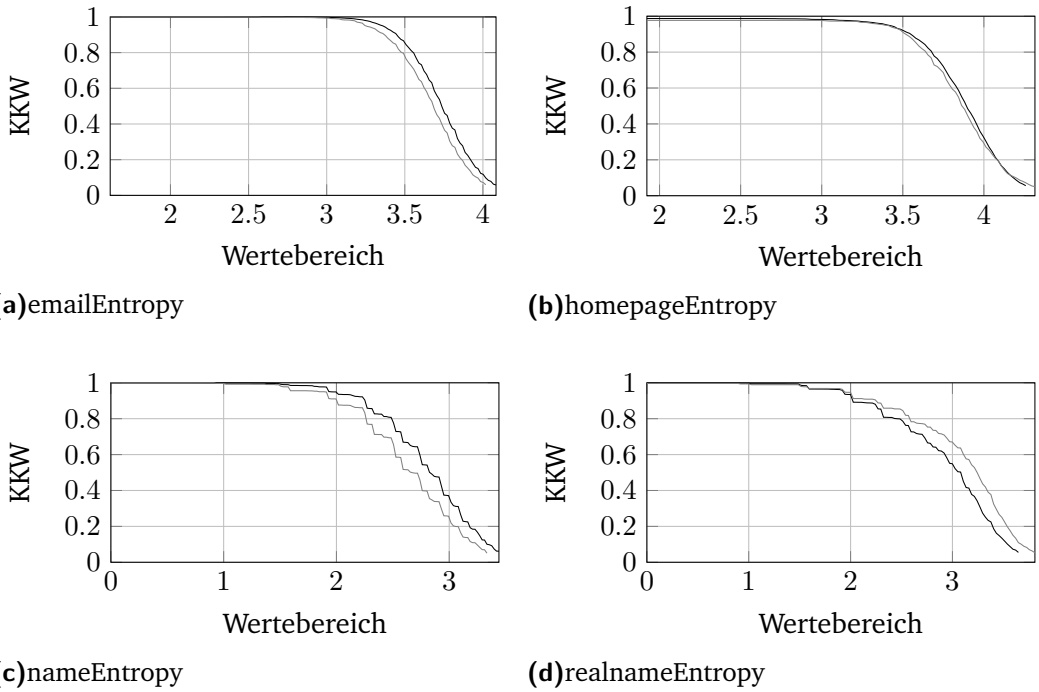
Feature Name	IG	Chi-Square
$nameInfoSupr$	0,002410	814,95
$nameLength$	0,002678	937,46
$nameNumDigit$	0,003216	713,37
$nameNumMaxTimesLetterRep$	0,000357	95,13
$nameNumStartingDigits$	0,000061	44,38
$nameNumUniqueAlphLetters$	0,000651	197,86
$namePropDigit$	0,003584	789,48
$namePropStartingDigits$	0,000000	0,00
$realname1$	0,000256	64,05
$realname2$	0,000098	24,84
$realname3$	0,000059	16,40
$realnameDigit$	0,000092	20,58
$realnameEntropy$	0,000509	136,61
$realnameInfoSupr$	0,000621	171,03
$realnameLength$	0,000595	164,09
$realnameNumDigit$	0,000092	20,58
$realnameNumMaxTimesLetterRep$	0,000212	58,57
$realnameNumStartingDigits$	0,000000	0,00
$realnameNumUniqueAlphLetters$	0,000379	101,26
$realnameParts$	0,000298	76,28
$realnamePropDigit$	0,000100	21,96
$realnamePropStartingDigits$	0,000000	0,00



(a)emailDigit **(b)**homepageDigit

(c)nameDigit **(d)**realnameDigit

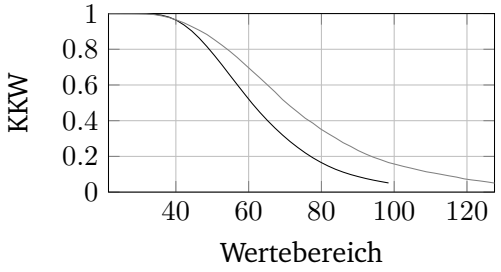
Abb. A.1.: Werte-Verteilungen des Meta-Features *digit*



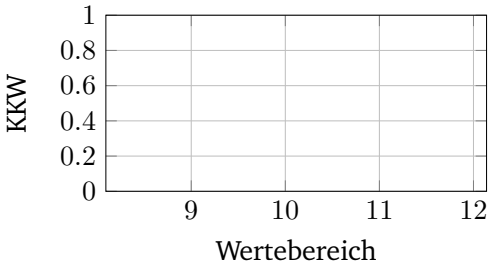
(a)emailEntropy **(b)**homepageEntropy

(c)nameEntropy **(d)**realnameEntropy

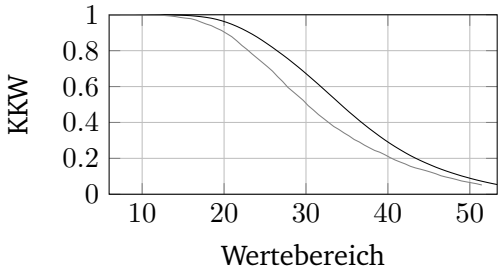
Abb. A.2.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features *entropy*



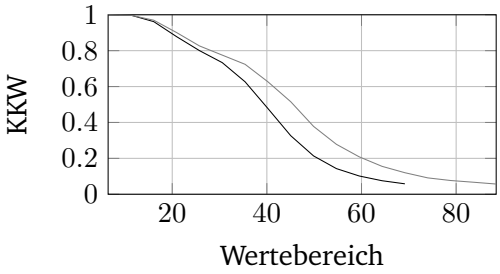
(a)emailInfoSupr



(b)homepageInfoSupr

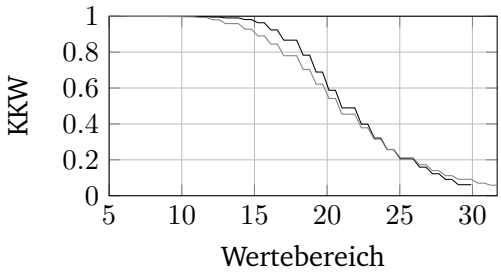


(c)nameInfoSupr

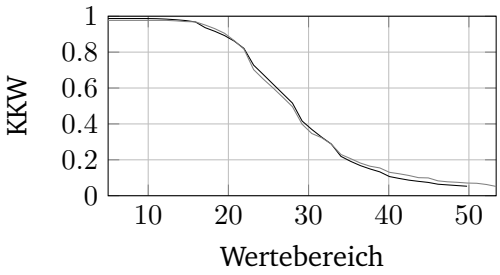


(d)realnameInfoSupr

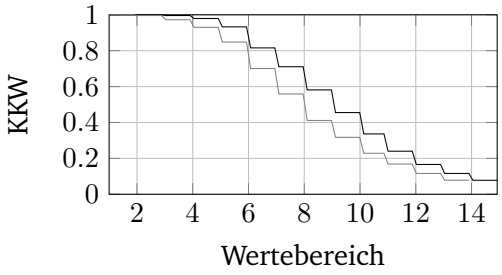
Abb. A.3.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $infoSupr$



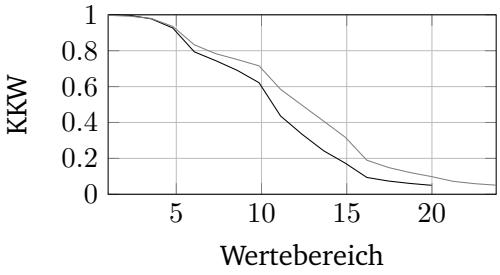
(a)emailLength



(b)homepageLength

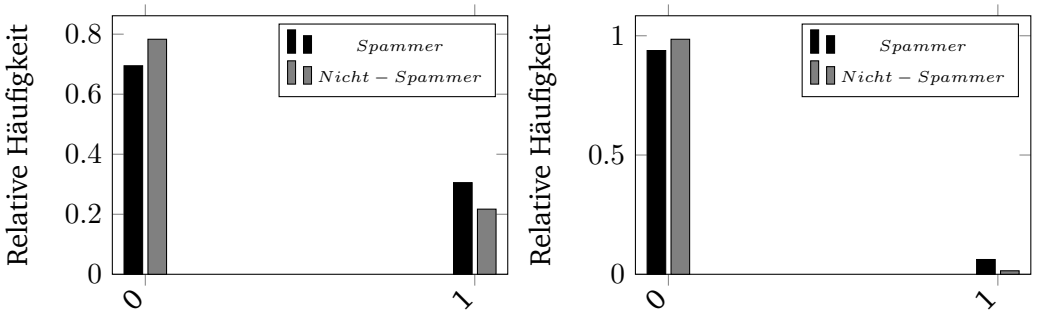


(c)nameLength



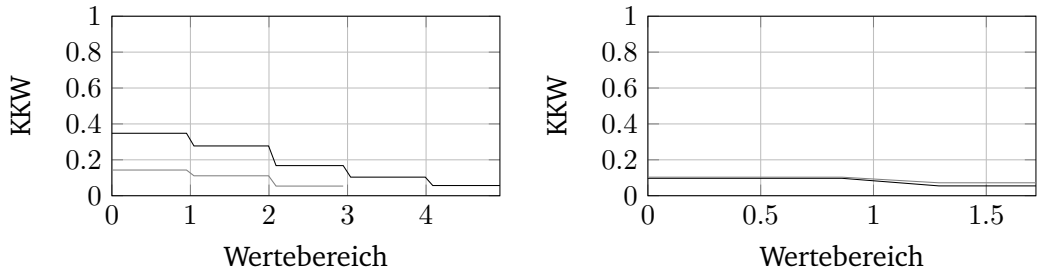
(d)realnameLength

Abb. A.4.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $length$

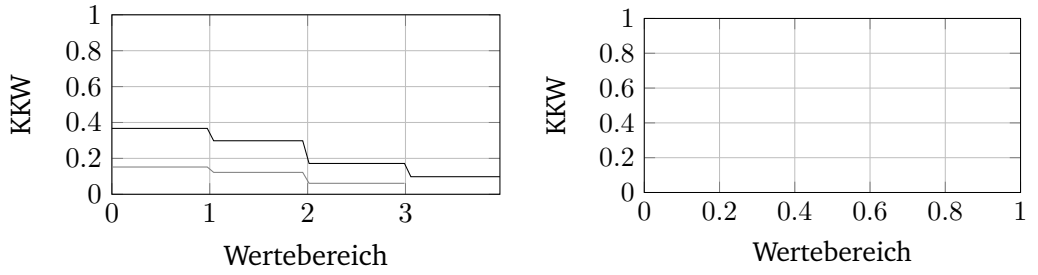


(a)nameEqualMail **(b)**nameEqualMailEqualRealnname

Abb. A.5.: Werte-Verteilungen der Feature a : $nameEqualMail$ & $nameEqualMailEqualRealnname$

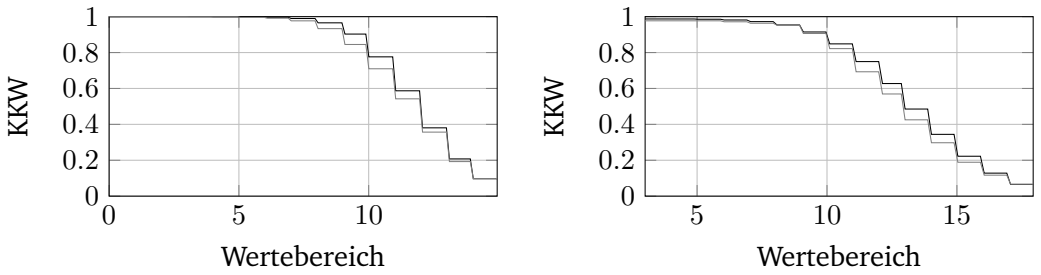


(a)emailNumDigit **(b)**homepageNumDigit

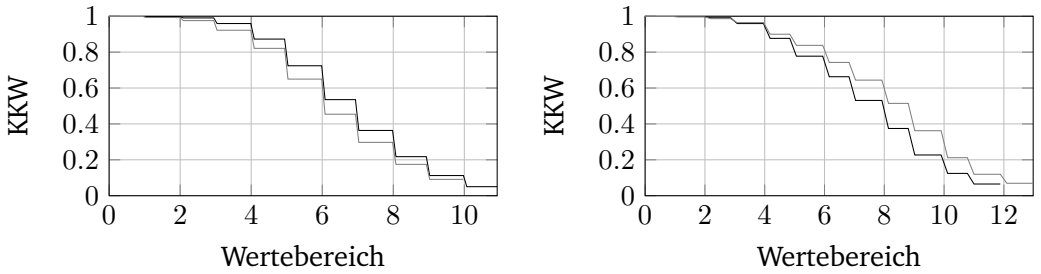


(c)nameNumDigit **(d)**realnameNumDigit

Abb. A.6.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $numDigit$

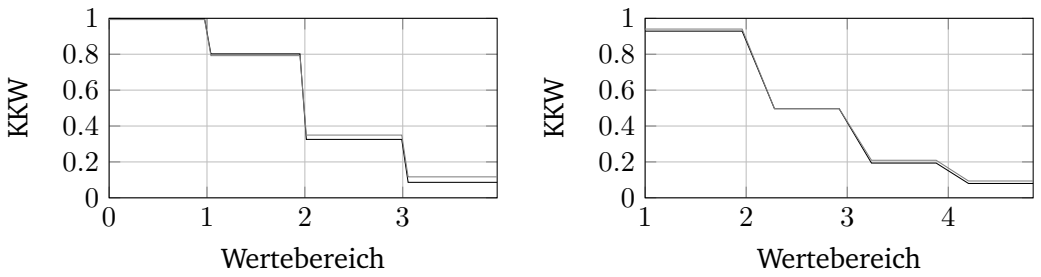


(a)emailNumUniqueAlphLetters **(b)**homepageNumUniqueAlphLetters

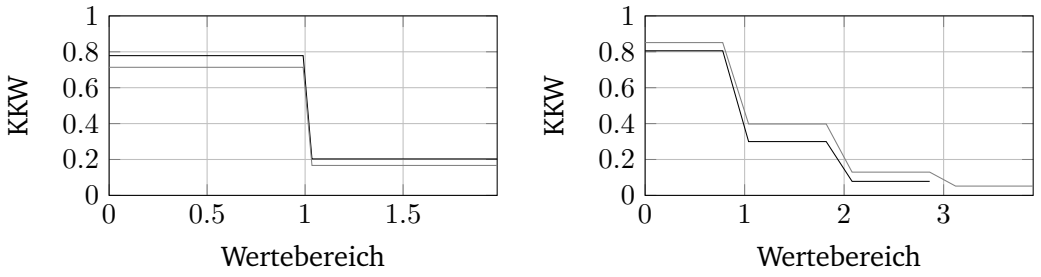


(c)nameNumUniqueAlphLetters **(d)**realnameNumUniqueAlphLetters

Abb. A.7.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $numUniqueAlphLetters$

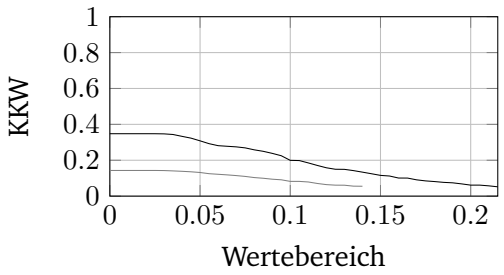


(a)emailNumMaxTimesLetterRep **(b)**homepageNumMaxTimesLetterRep

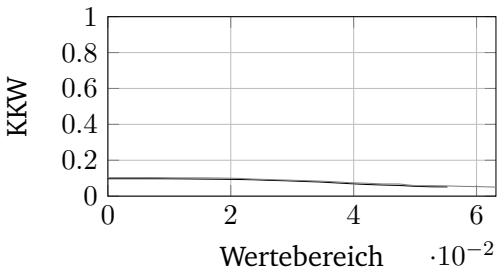


(c)nameNumMaxTimesLetterRep **(d)**realnameNumMaxTimesLetterRep

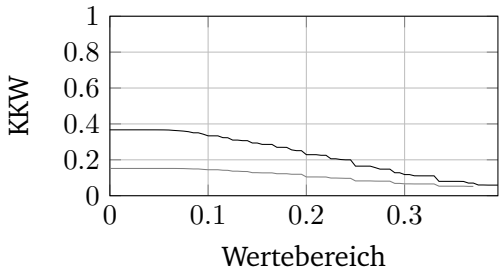
Abb. A.8.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $numMaxTimesLetterRep$



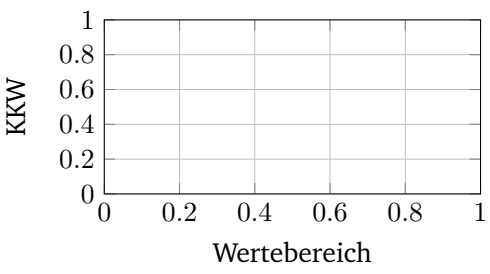
(a)emailPropDigit



(b)homepagePropDigit

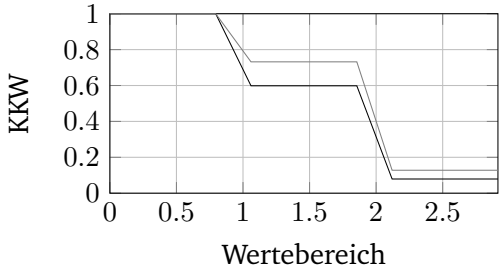


(c)namePropDigit



(d)realnamePropDigit

Abb. A.9.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propDigit$



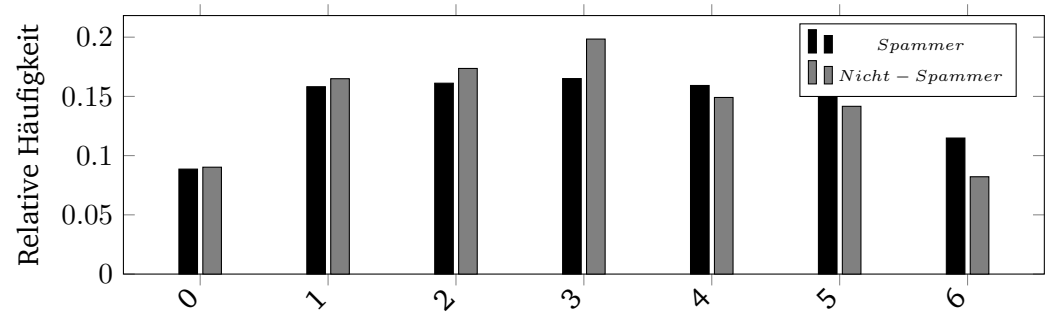
(a)realnameParts

Abb. A.10.: Komplementäre Empirische Verteilungsfunktionen des Feature $realnameParts$

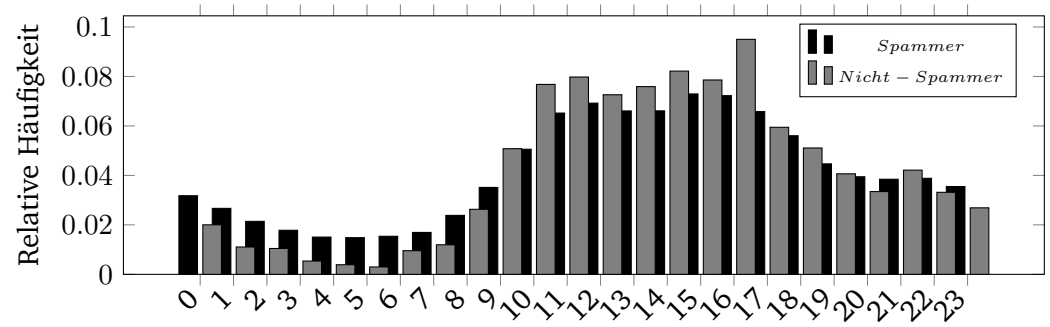
A.2 Umwelt-basierte Features

Tab. A.3.: Umwelt-basierte Features

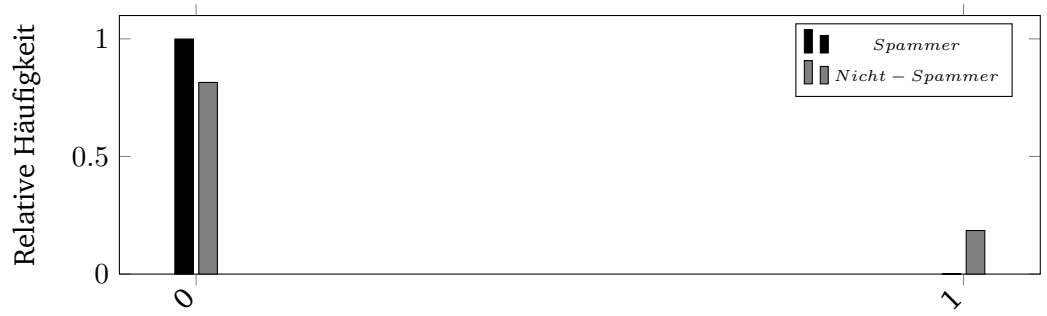
Feature Name	IG	Chi-Square
$country$	0,039396	25170,73
$regDay$	0,000231	57,27
$regHour$	0,001234	281,02
$uniMail$	0,018293	27720,79



(a)regDay



(b)regHour



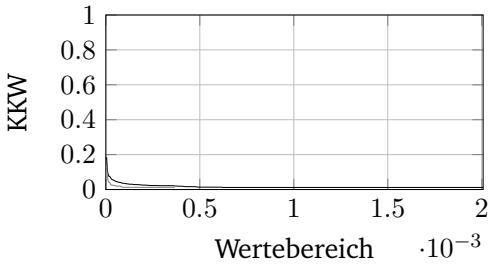
(c)uniMail

Abb. A.11.: Werte-Verteilungen der Feature $regDay$, $regHour$ & $uniMail$

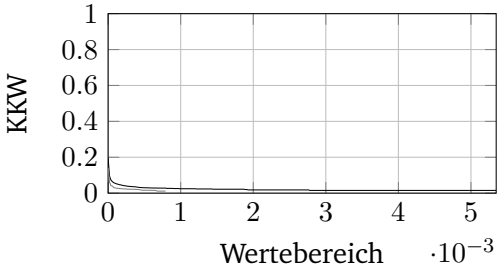
A.3 Populations-basierte Features

Tab. A.4.: Populations-basierte Features

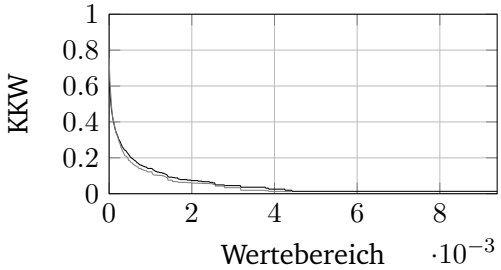
Feature Name	IG	Chi-Square
$domainCount$	0,000013	3,26
$domainRatio$	0,000035	12,44
$emailCount$	0,000726	200,03
$emailRatio$	0,000000	0,00
$firstnameCount$	0,000194	53,43
$firstnameRatio$	0,004680	2658,22
$ipRatio$	0,000479	319,86
$lastnameCount$	0,000643	166,59
$lastnameRatio$	0,000625	291,49
$nameCount$	0,000523	118,39
$nameRatio$	0,000156	67,51
$realnameCount$	0,000162	39,80
$realnameRatio$	0,000273	113,41
$spamIP$	0,012551	2405,35
$tldCount$	0,002254	1040,96
$tldRatio$	0,002564	1264,06



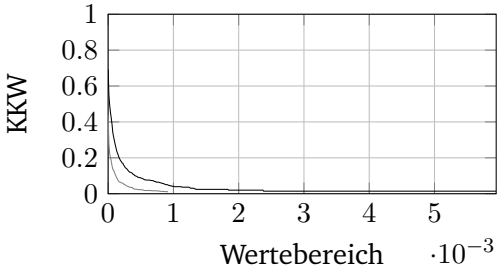
(a)domainCount



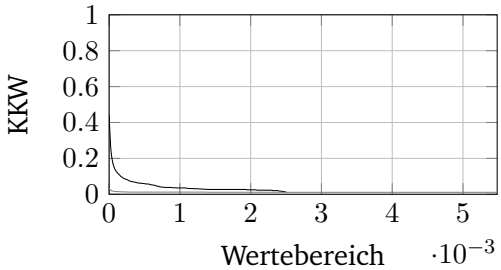
(b)emailCount



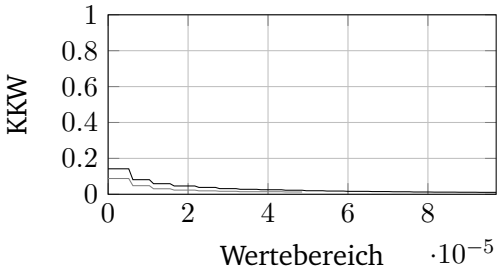
(c)firstnameCount



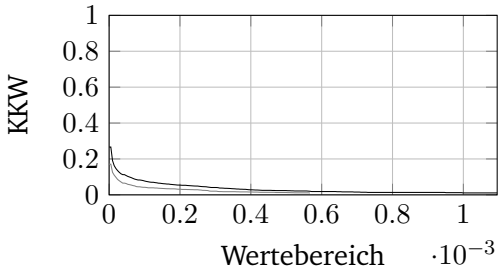
(d)lastnameCount



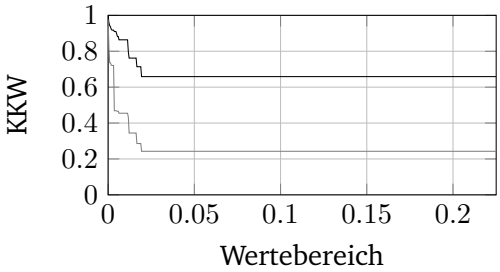
(e)spamIP



(f)nameCount

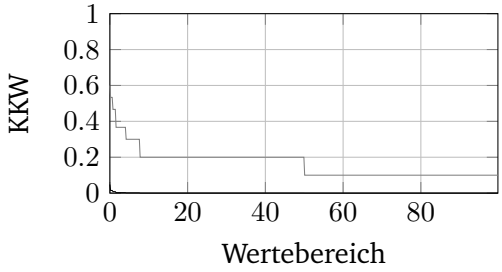


(g)realnameCount

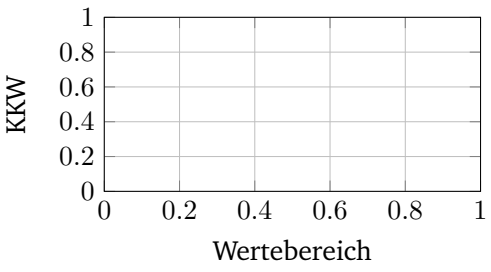


(h)tldCount

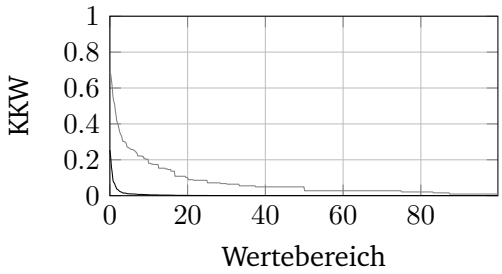
Abb. A.12.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features *count*



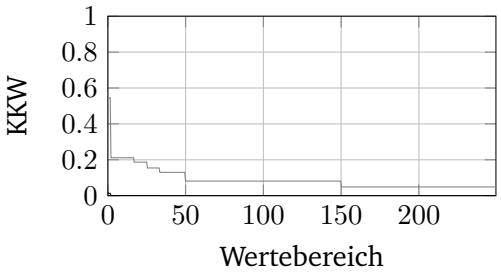
(a)domainRatio



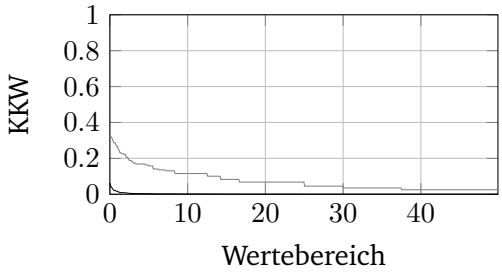
(b)emailRatio



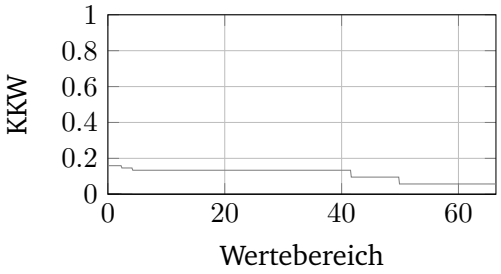
(c)firstnameRatio



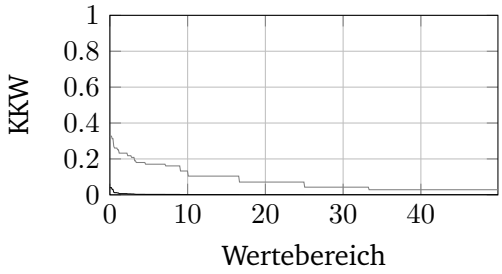
(d)ipRatio



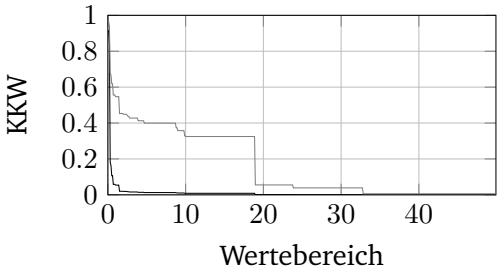
(e)lastnameRatio



(f)nameRatio



(g)realnameRatio



(h)tldRatio

Abb. A.13.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $ratio$ 1 / 2

A.4 Tastatur-basierte Features

Tab. A.5.: Tastatur-basierte Features 1 / 3

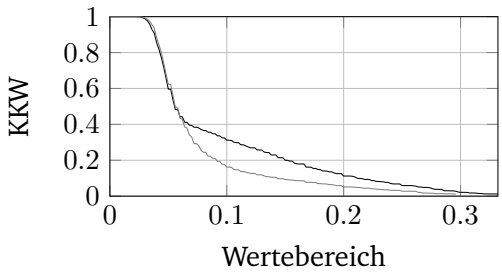
Feature Name	IG	Chi-Square
$emailDistanceOnKeyboardDvorak$	0,014066	13529,95
$emailDistanceOnKeyboardQwertz$	0,015035	14635,66
$emailProp1RowDvorak$	0,003570	1002,95
$emailProp1RowQwertz$	0,003279	873,51
$emailProp2RowDvorak$	0,001168	443,53
$emailProp2RowQwertz$	0,001126	300,53
$emailProp3RowDvorak$	0,003166	1072,38
$emailProp3RowQwertz$	0,000479	155,17
$emailProp4RowDvorak$	0,001174	485,50
$emailProp4RowQwertz$	0,001270	572,31
$emailPropForefingerLeftDvorak$	0,000195	70,67
$emailPropForefingerLeftQwertz$	0,000293	80,73
$emailPropForefingerRightDvorak$	0,000639	679,41
$emailPropForefingerRightQwertz$	0,001209	623,59
$emailPropLittleFingerLeftDvorak$	0,002127	728,68
$emailPropLittleFingerLeftQwertz$	0,003601	1215,71
$emailPropLittleFingerRightDvorak$	0,000988	267,91
$emailPropLittleFingerRightQwertz$	0,001557	537,94
$emailPropMiddleFingerLeftDvorak$	0,010904	4981,48
$emailPropMiddleFingerLeftQwertz$	0,004582	1861,40
$emailPropMiddleFingerRightDvorak$	0,002581	1120,27
$emailPropMiddleFingerRightQwertz$	0,000454	125,81
$emailPropRingFingerLeftDvorak$	0,010617	5262,72
$emailPropRingFingerLeftQwertz$	0,000820	267,61
$emailPropRingFingerRightDvorak$	0,000674	183,67
$emailPropRingFingerRightQwertz$	0,003595	1459,54
$emailPropSameFingerAsPrevDvorak$	0,001150	331,15
$emailPropSameFingerAsPrevQwertz$	0,000525	129,84
$emailPropSameHandAsPrevDvorak$	0,000349	159,44
$emailPropSameHandAsPrevQwertz$	0,000238	61,91
$homepageDistanceOnKeyboardDvorak$	0,000457	388,06
$homepageDistanceOnKeyboardQwertz$	0,000000	0,00
$homepageProp1RowDvorak$	0,000057	16,46
$homepageProp1RowQwertz$	0,000000	0,00
$homepageProp2RowDvorak$	0,000000	0,00
$homepageProp2RowQwertz$	0,000000	0,00
$homepageProp3RowDvorak$	0,000000	0,00
$homepageProp3RowQwertz$	0,000035	8,58

Tab. A.6.: Tastatur-basierte Features 2 / 3

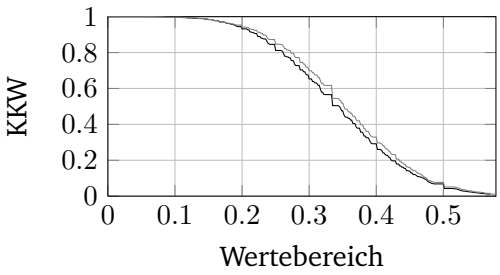
Feature Name	IG	Chi-Square
$homepageProp4RowDvorak$	0,000023	5,75
$homepageProp4RowQwertz$	0,000146	38,66
$homepagePropForefingerLeftDvorak$	0,000022	5,73
$homepagePropForefingerLeftQwertz$	0,000000	0,00
$homepagePropForefingerRightDvorak$	0,000009	2,07
$homepagePropForefingerRightQwertz$	0,000000	0,00
$homepagePropLittleFingerLeftDvorak$	0,000000	0,00
$homepagePropLittleFingerLeftQwertz$	0,000015	3,77
$homepagePropLittleFingerRightDvorak$	0,000000	0,00
$homepagePropLittleFingerRightQwertz$	0,000029	7,34
$homepagePropMiddleFingerLeftDvorak$	0,000186	49,95
$homepagePropMiddleFingerLeftQwertz$	0,000000	0,00
$homepagePropMiddleFingerRightDvorak$	0,000068	18,10
$homepagePropMiddleFingerRightQwertz$	0,000026	6,93
$homepagePropRingFingerLeftDvorak$	0,000311	88,19
$homepagePropRingFingerLeftQwertz$	0,000000	0,00
$homepagePropRingFingerRightDvorak$	0,000000	0,00
$homepagePropRingFingerRightQwertz$	0,000017	4,47
$homepagePropSameFingerAsPrevDvorak$	0,000000	0,00
$homepagePropSameFingerAsPrevQwertz$	0,000045	11,85
$homepagePropSameHandAsPrevDvorak$	0,000018	4,61
$homepagePropSameHandAsPrevQwertz$	0,000000	0,00
$nameDistanceOnKeyboardDvorak$	0,001961	425,23
$nameDistanceOnKeyboardQwertz$	0,001961	425,26
$nameProp1RowDvorak$	0,003560	784,83
$nameProp1RowQwertz$	0,003584	789,54
$nameProp2RowDvorak$	0,000397	94,26
$nameProp2RowQwertz$	0,000477	118,54
$nameProp3RowDvorak$	0,001001	240,83
$nameProp3RowQwertz$	0,000544	127,91
$nameProp4RowDvorak$	0,000454	132,51
$nameProp4RowQwertz$	0,000685	157,88
$namePropForefingerLeftDvorak$	0,000550	137,99
$namePropForefingerLeftQwertz$	0,000223	56,26
$namePropForefingerRightDvorak$	0,000059	15,77
$namePropForefingerRightQwertz$	0,000581	172,92
$namePropLittleFingerLeftDvorak$	0,000676	167,14
$namePropLittleFingerLeftQwertz$	0,001088	274,66
$namePropLittleFingerRightDvorak$	0,000530	128,76
$namePropLittleFingerRightQwertz$	0,000336	110,90
$namePropMiddleFingerLeftDvorak$	0,000846	220,81

Tab. A.7.: Tastatur-basierte Features 3 / 3

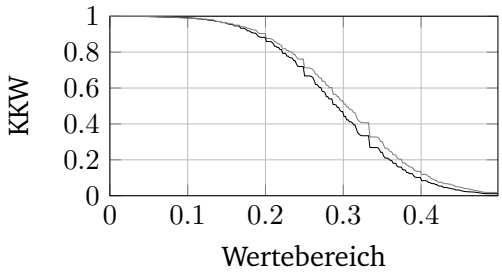
Feature Name	IG	Chi-Square
$namePropMiddleFingerLeftQwertz$	0,000521	127,30
$namePropMiddleFingerRightDvorak$	0,000380	91,87
$namePropMiddleFingerRightQwertz$	0,000447	113,52
$namePropRingFingerLeftDvorak$	0,000958	226,15
$namePropRingFingerLeftQwertz$	0,000635	152,56
$namePropRingFingerRightDvorak$	0,000714	166,66
$namePropRingFingerRightQwertz$	0,000333	81,32
$namePropSameFingerAsPrevDvorak$	0,001045	260,48
$namePropSameFingerAsPrevQwertz$	0,000937	240,69
$namePropSameHandAsPrevDvorak$	0,000443	130,49
$namePropSameHandAsPrevQwertz$	0,000543	158,71
$realnameDistanceOnKeyboardDvorak$	0,000172	49,20
$realnameDistanceOnKeyboardQwertz$	0,000174	49,80
$realnameProp1RowDvorak$	0,000086	18,60
$realnameProp1RowQwertz$	0,000086	18,56
$realnameProp2RowDvorak$	0,000101	27,61
$realnameProp2RowQwertz$	0,000081	20,04
$realnameProp3RowDvorak$	0,000050	12,05
$realnameProp3RowQwertz$	0,000041	9,79
$realnameProp4RowDvorak$	0,000056	13,49
$realnameProp4RowQwertz$	0,000059	14,75
$realnamePropForefingerLeftDvorak$	0,000000	0,00
$realnamePropForefingerLeftQwertz$	0,000126	33,75
$realnamePropForefingerRightDvorak$	0,000137	36,60
$realnamePropForefingerRightQwertz$	0,000065	15,36
$realnamePropLittleFingerLeftDvorak$	0,000200	56,80
$realnamePropLittleFingerLeftQwertz$	0,000294	84,46
$realnamePropLittleFingerRightDvorak$	0,000177	48,00
$realnamePropLittleFingerRightQwertz$	0,000208	60,10
$realnamePropMiddleFingerLeftDvorak$	0,000181	50,26
$realnamePropMiddleFingerLeftQwertz$	0,000131	34,56
$realnamePropMiddleFingerRightDvorak$	0,000198	55,08
$realnamePropMiddleFingerRightQwertz$	0,000160	43,17
$realnamePropRingFingerLeftDvorak$	0,000237	65,22
$realnamePropRingFingerLeftQwertz$	0,000086	20,93
$realnamePropRingFingerRightDvorak$	0,000072	18,01
$realnamePropRingFingerRightQwertz$	0,000036	9,06
$realnamePropSameFingerAsPrevDvorak$	0,000292	85,61
$realnamePropSameFingerAsPrevQwertz$	0,000158	43,20
$realnamePropSameHandAsPrevDvorak$	0,000048	11,69
$realnamePropSameHandAsPrevQwertz$	0,000068	16,55



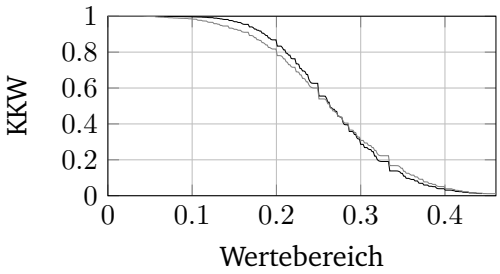
(a)emailProp1RowQwertz



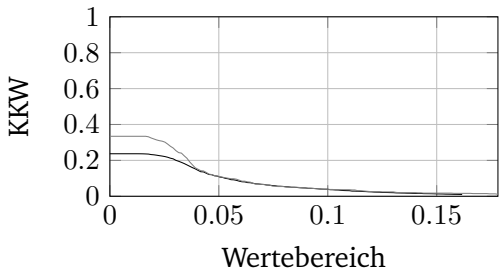
(b)emailProp2RowQwertz



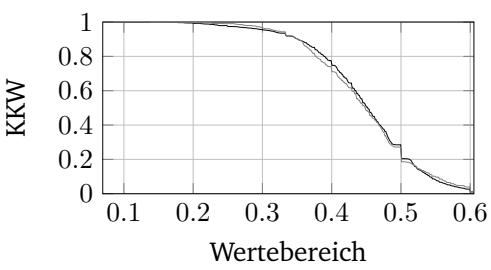
(c)emailProp3RowQwertz



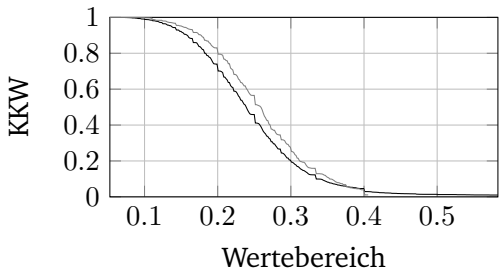
(d)emailProp4RowQwertz



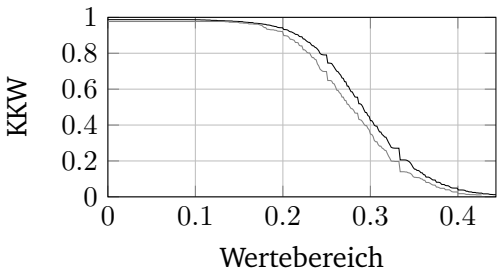
(e)homepageProp1RowQwertz



(f)homepageProp2RowQwertz

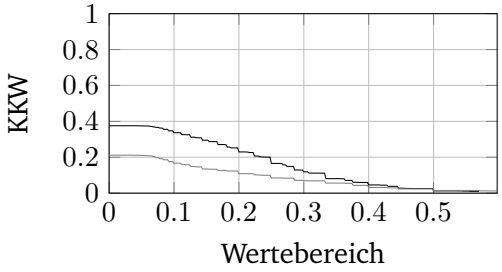


(g)homepageProp3RowQwertz

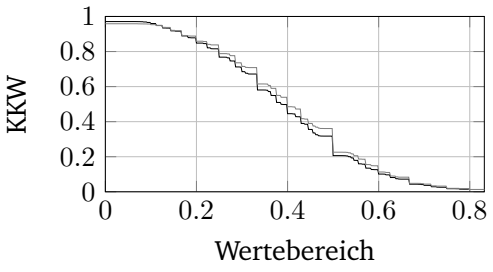


(h)homepageProp4RowQwertz

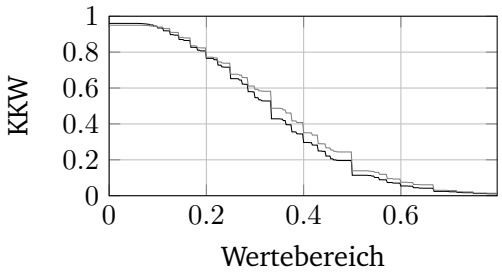
Abb. A.14.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$



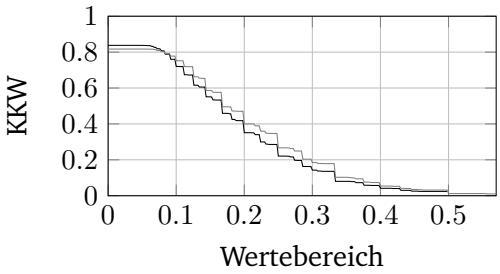
(a)namePrep1RowQwertz



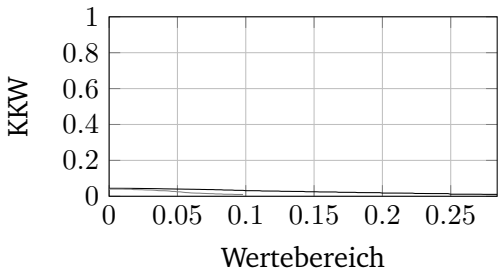
(b)namePrep2RowQwertz



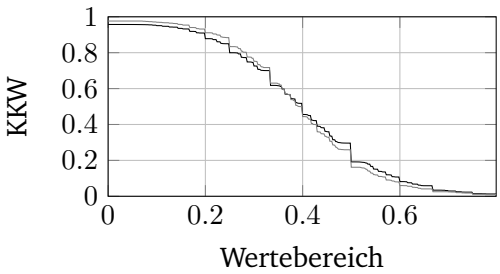
(c)namePrep3RowQwertz



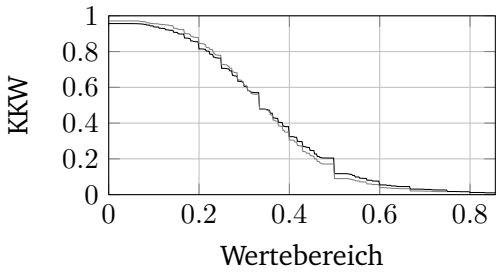
(d)namePrep4RowQwertz



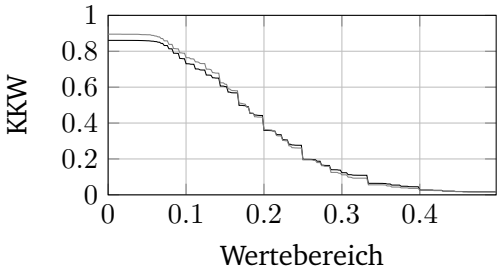
(e)realnamePrep1RowQwertz



(f)realnamePrep2RowQwertz

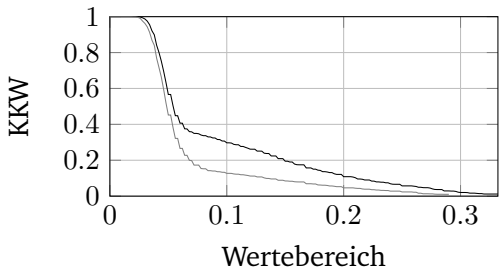


(g)realnamePrep3RowQwertz

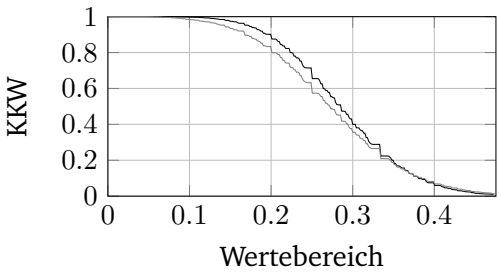


(h)realnamePrep4RowQwertz

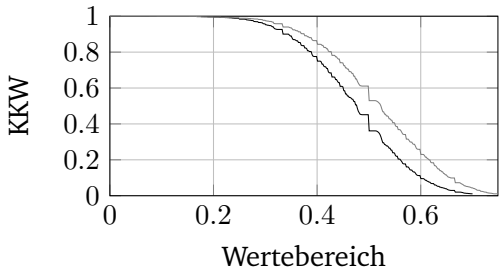
Abb. A.15.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$



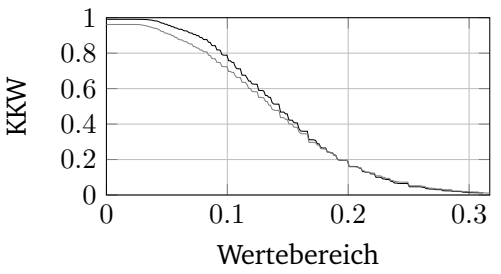
(a)emailProp1RowDvorak



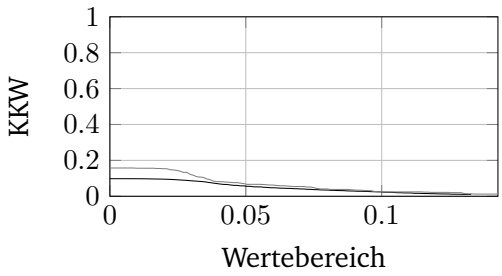
(b)emailProp2RowDvorak



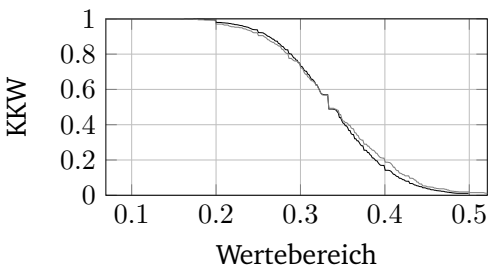
(c)emailProp3RowDvorak



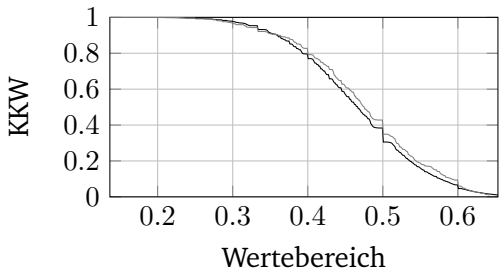
(d)emailProp4RowDvorak



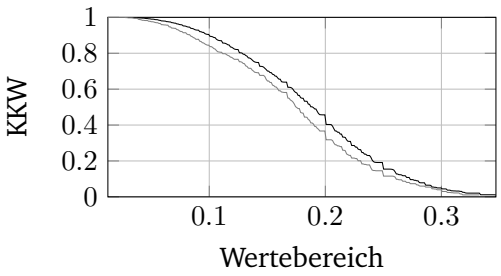
(e)homepageProp1RowDvorak



(f)homepageProp2RowDvorak

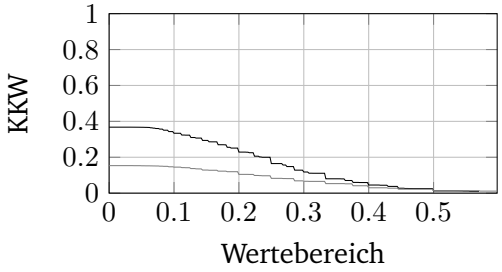


(g)homepageProp3RowDvorak

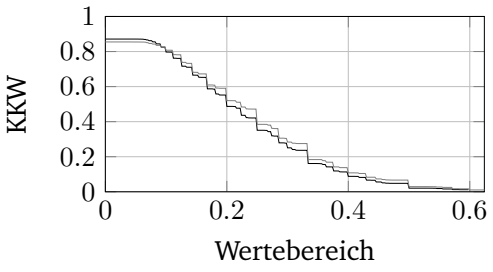


(h)homepageProp4RowDvorak

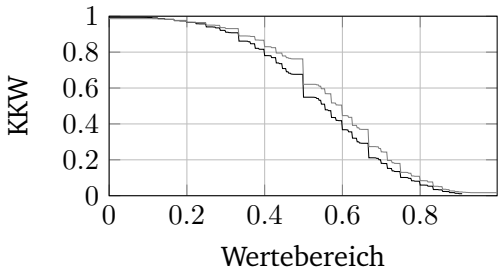
Abb. A.16.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$



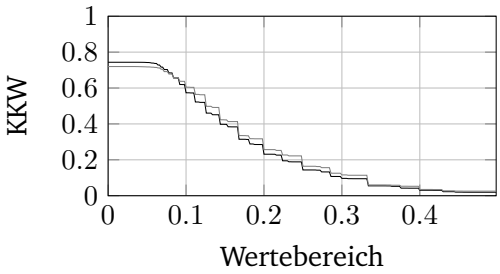
(a)namePrep1RowDvorak



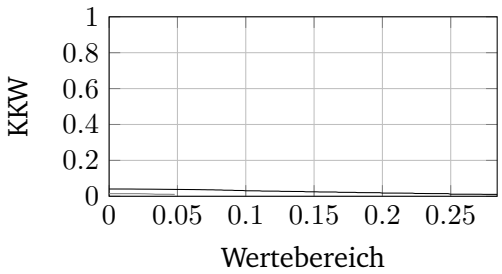
(b)namePrep2RowDvorak



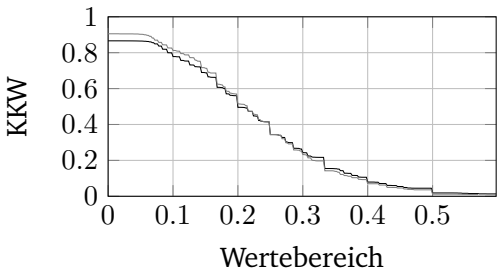
(c)namePrep3RowDvorak



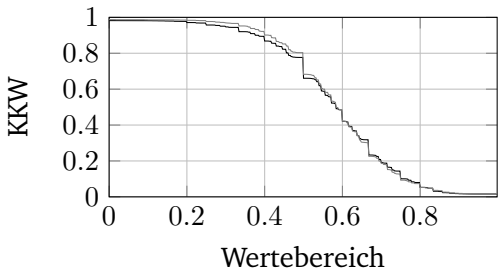
(d)namePrep4RowDvorak



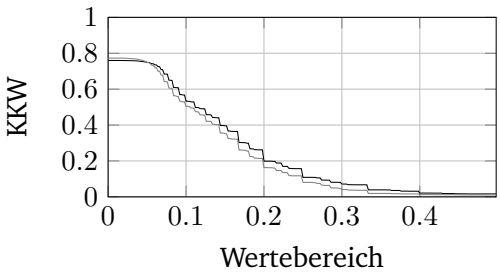
(e)realnamePrep1RowDvorak



(f)realnamePrep2RowDvorak

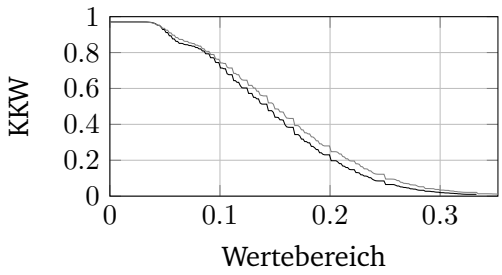


(g)realnamePrep3RowDvorak

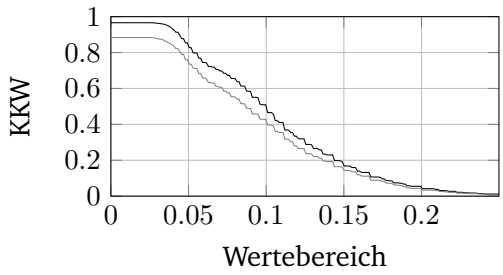


(h)realnamePrep4RowDvorak

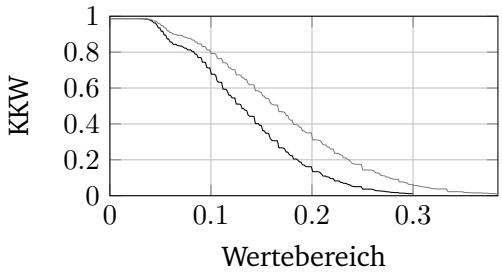
Abb. A.17.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$



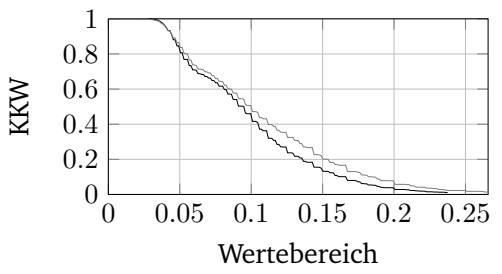
(a)emailPropForefingerLeftQwertz



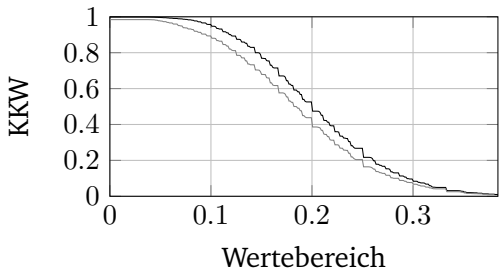
(b)emailPropLittleFingerLeftQwertz



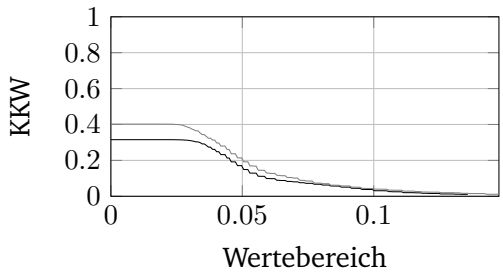
(c)emailPropMiddleFingerLeftQwertz



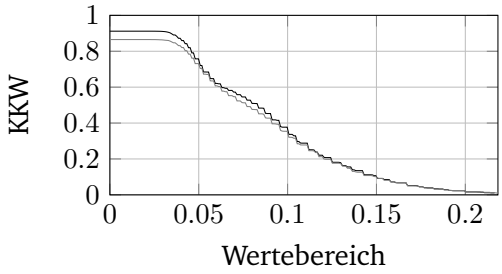
(d)emailPropRingFingerLeftQwertz



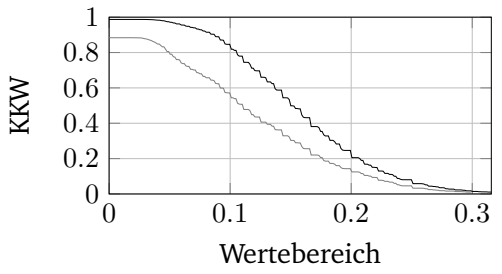
(e)emailPropForefingerRightQwertz



(f)emailPropLittleFingerRightQwertz

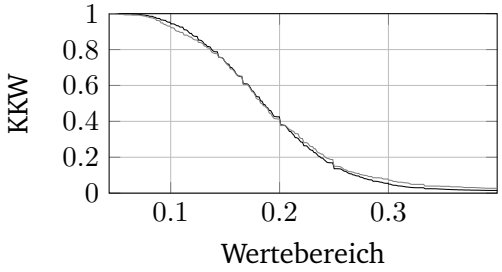


(g)emailPropMiddleFingerRightQwertz

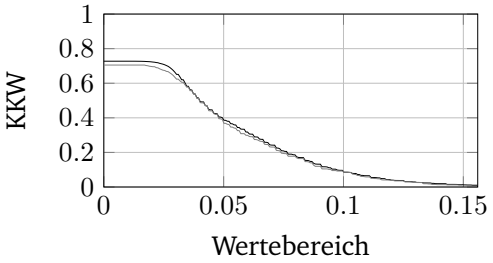


(h)emailPropRingFingerRightQwertz

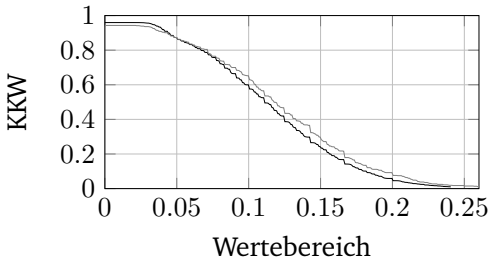
Abb. A.18.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$



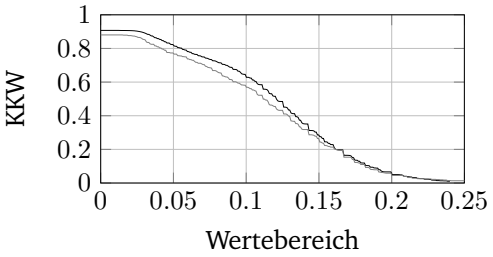
(a)homepagePropForefingerLeftQwertz



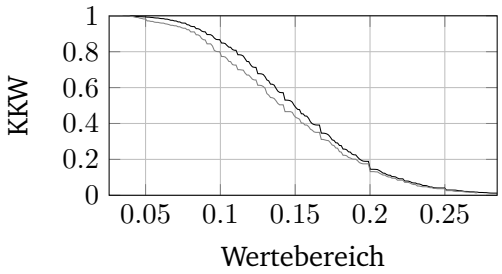
(b)homepagePropLittleFingerLeftQwertz



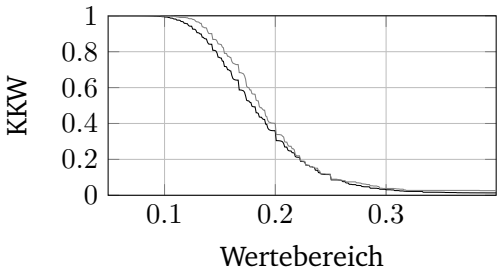
(c)homepagePropMiddleFingerLeftQwertz



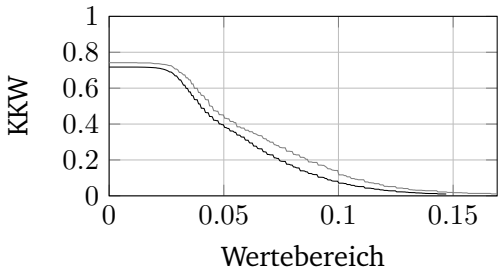
(d)homepagePropRingFingerLeftQwertz



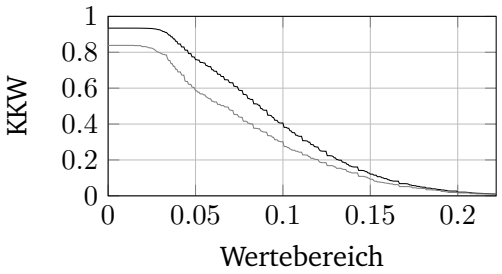
(e)homepagePropForefingerRightQwertz



(f)homepagePropLittleFingerRightQwertz

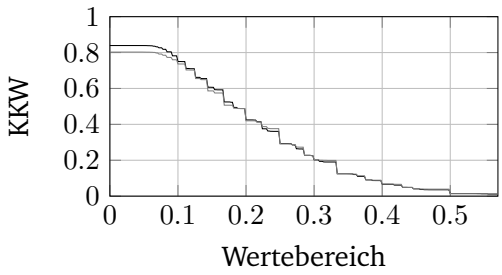


(g)homepagePropMiddleFingerRightQwertz

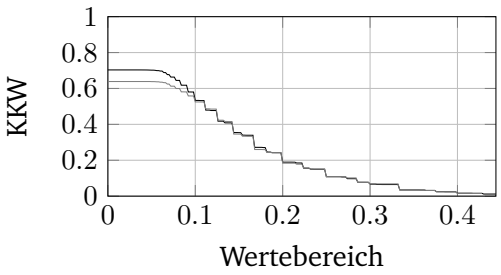


(h)homepagePropRingFingerRightQwertz

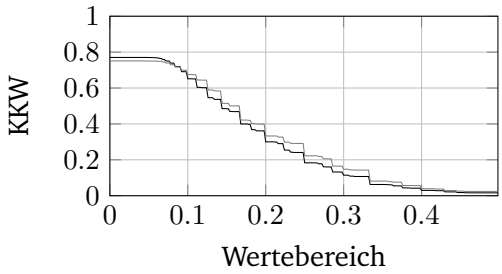
Abb. A.19.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$



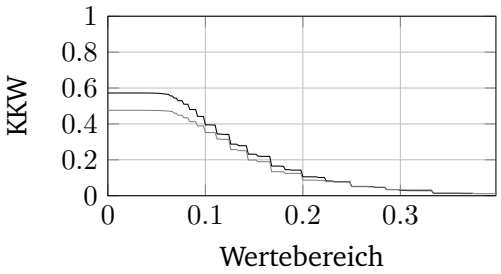
(a)namePropForefingerLeftQwertz



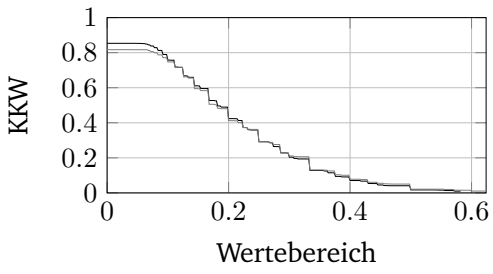
(b)namePropLittleFingerLeftQwertz



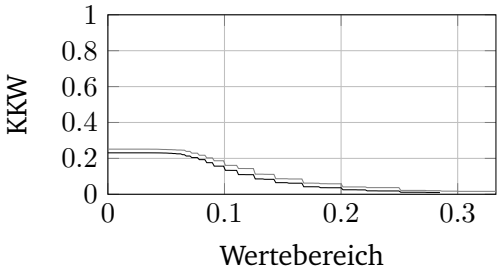
(c)namePropMiddleFingerLeftQwertz



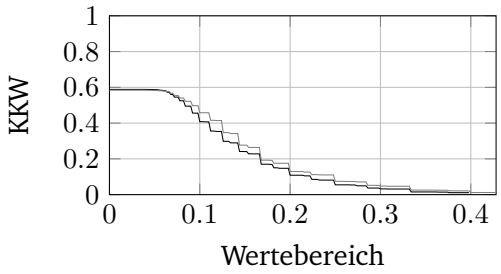
(d)namePropRingFingerLeftQwertz



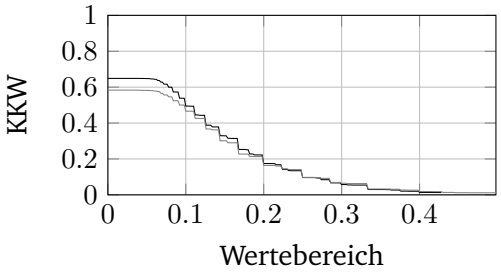
(e)namePropForefingerRightQwertz



(f)namePropLittleFingerRightQwertz

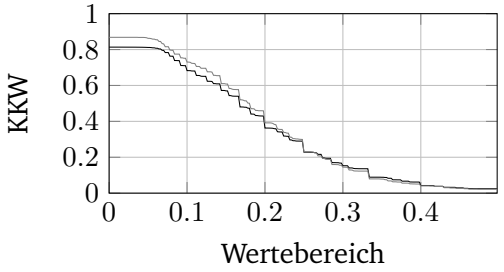


(g)namePropMiddleFingerRightQwertz

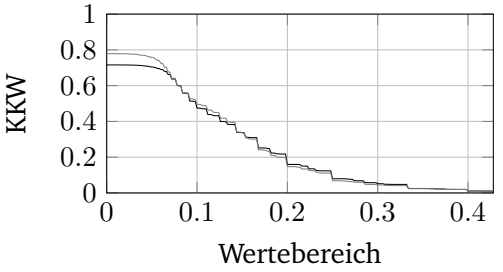


(h)namePropRingFingerRightQwertz

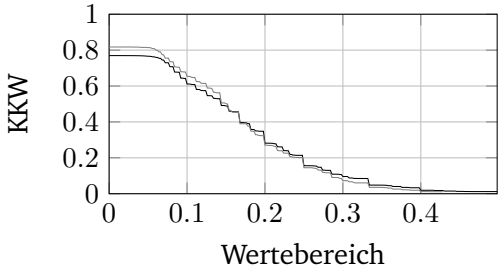
Abb. A.20.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$



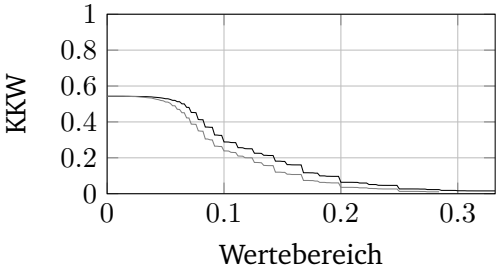
(a)realnamePropForefingerLeftQwertz



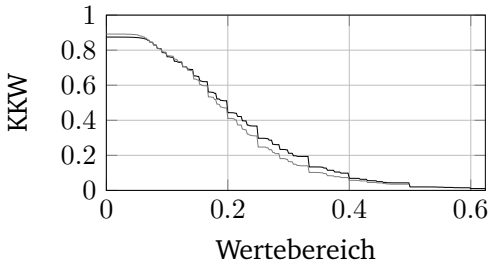
(b)realnamePropLittleFingerLeftQwertz



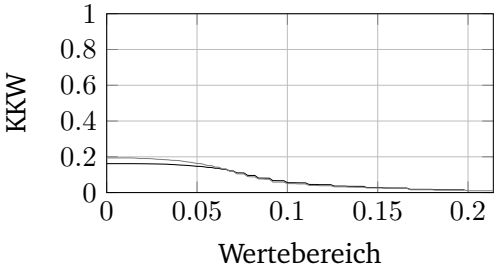
(c)realnamePropMiddleFingerLeftQwertz



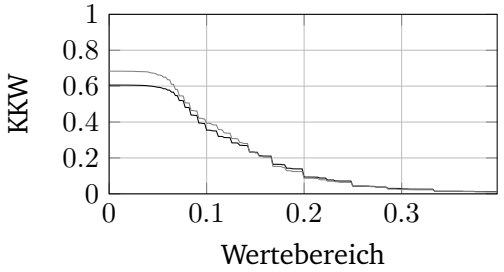
(d)realnamePropRingFingerLeftQwertz



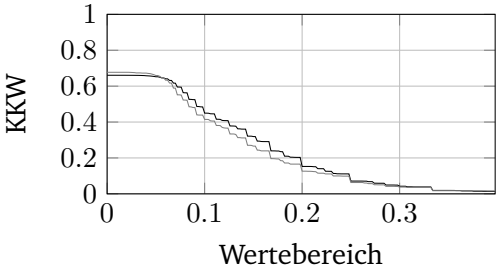
(e)realnamePropForefingerRightQwertz



(f)realnamePropLittleFingerRightQwertz

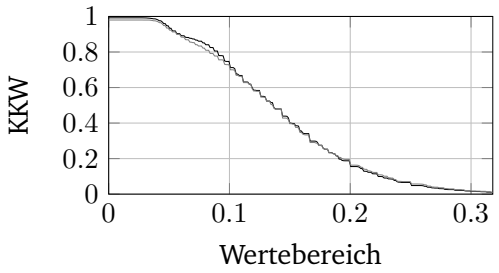


(g)realnamePropMiddleFingerRightQwertz

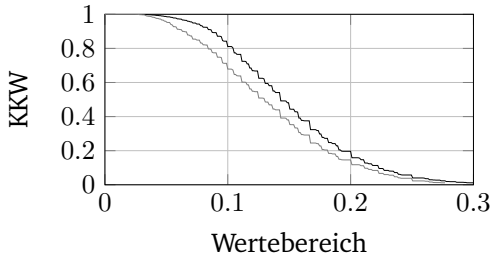


(h)realnamePropRingFingerRightQwertz

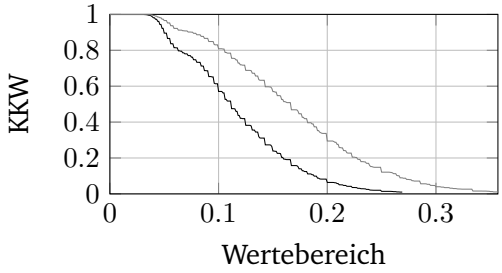
Abb. A.21.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$



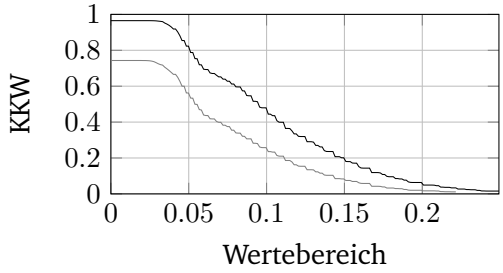
(a)emailPropForefingerLeftDvorak



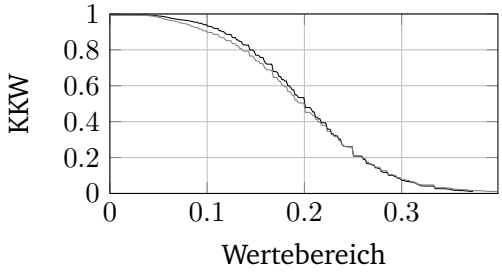
(b)emailPropLittleFingerLeftDvorak



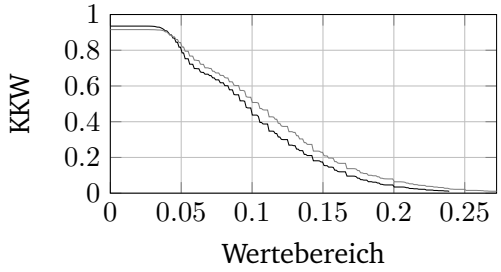
(c)emailPropMiddleFingerLeftDvorak



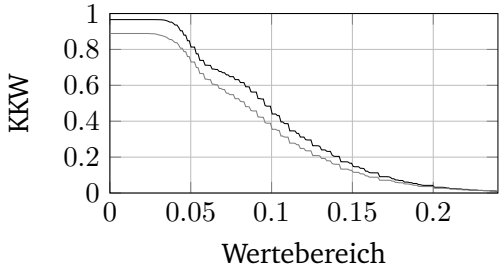
(d)emailPropRingFingerLeftDvorak



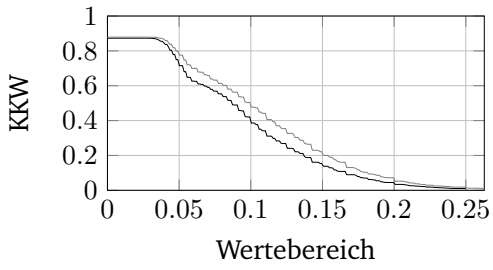
(e)emailPropForefingerRightDvorak



(f)emailPropLittleFingerRightDvorak

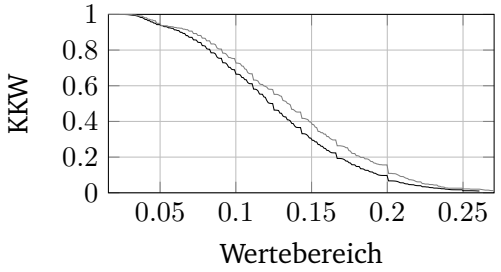


(g)emailPropMiddleFingerRightDvorak

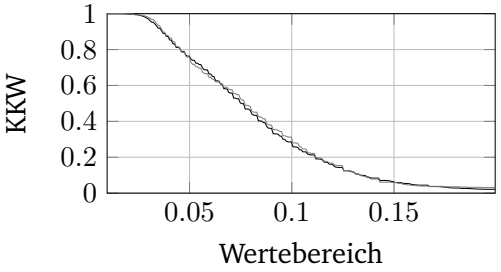


(h)emailPropRingFingerRightDvorak

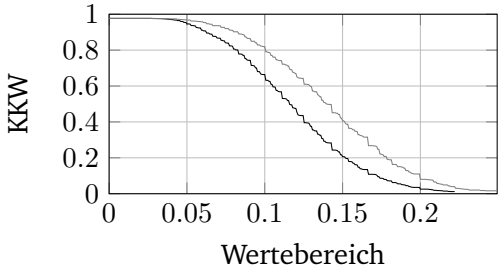
Abb. A.22.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$



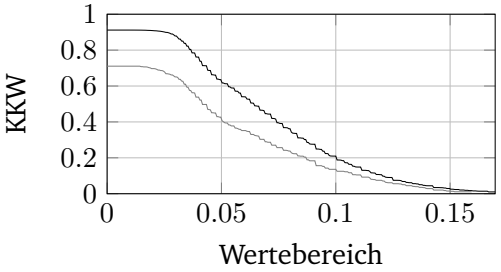
(a)homepagePropForefingerLeftDvorak



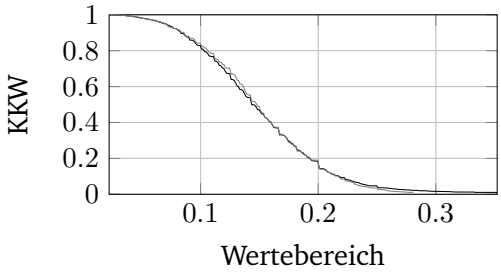
(b)homepagePropLittleFingerLeftDvorak



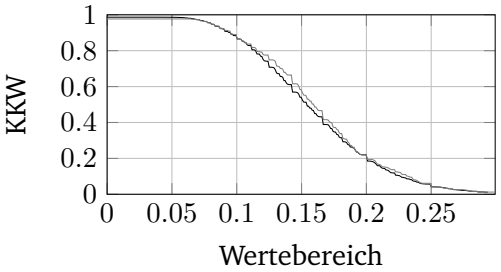
(c)homepagePropMiddleFingerLeftDvorak



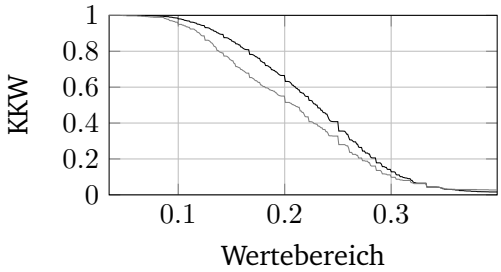
(d)homepagePropRingFingerLeftDvorak



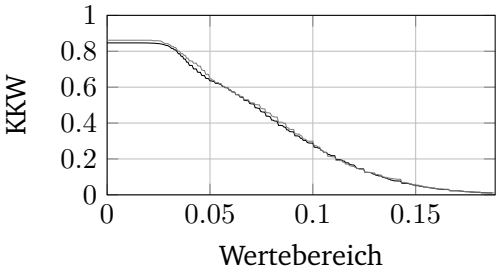
(e)homepagePropForefingerRightDvorak



(f)homepagePropLittleFingerRightDvorak

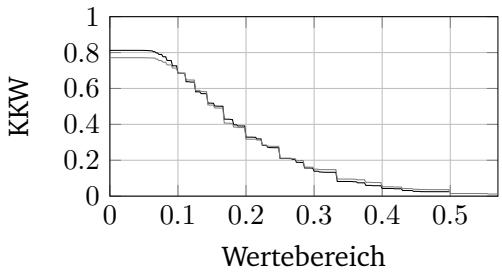


(g)homepagePropMiddleFingerRightDvorak

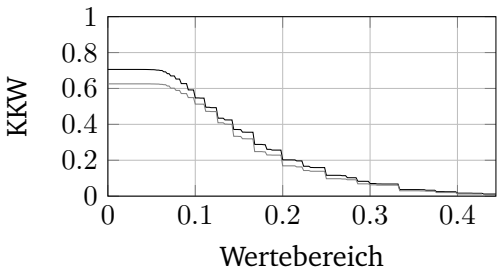


(h)homepagePropRingFingerRightDvorak

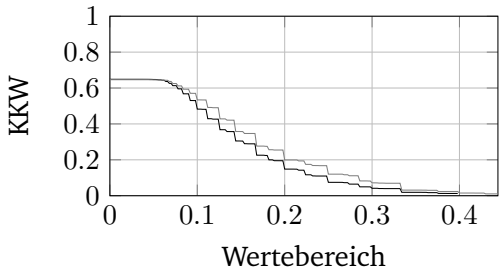
Abb. A.23.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$



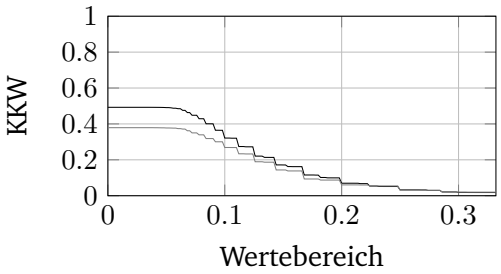
(a)namePropForefingerLeftDvorak



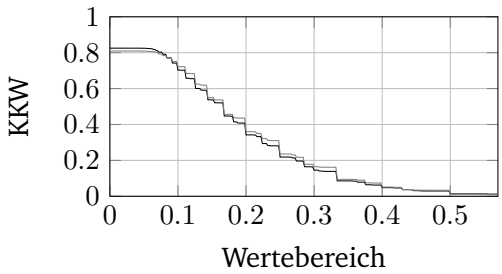
(b)namePropLittleFingerLeftDvorak



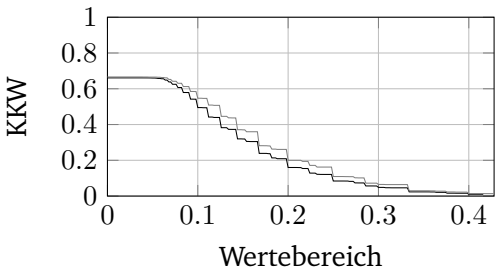
(c)namePropMiddleFingerLeftDvorak



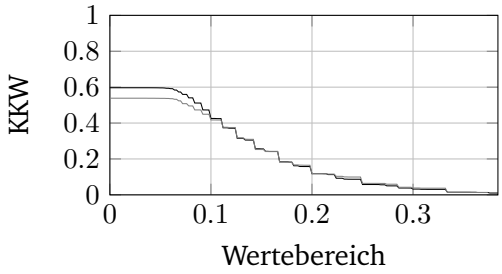
(d)namePropRingFingerLeftDvorak



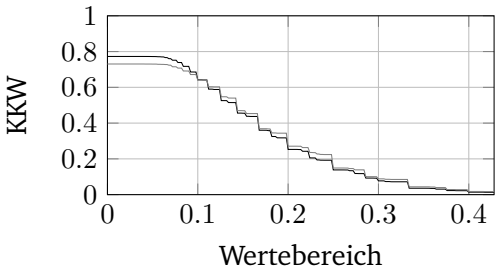
(e)namePropForefingerRightDvorak



(f)namePropLittleFingerRightDvorak

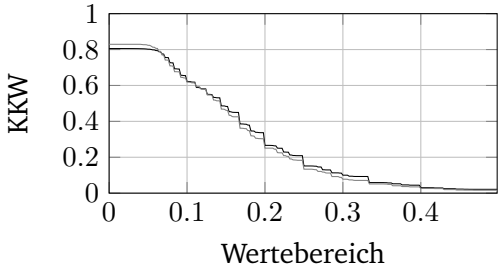


(g)namePropMiddleFingerRightDvorak

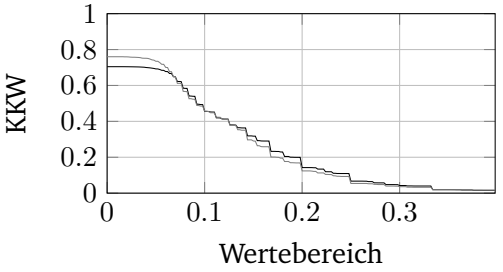


(h)namePropRingFingerRightDvorak

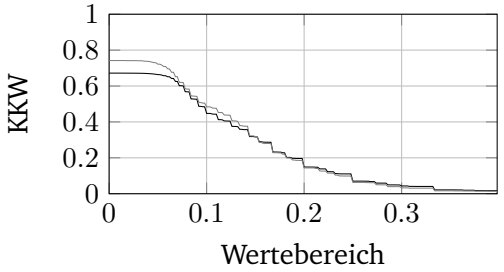
Abb. A.24.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$



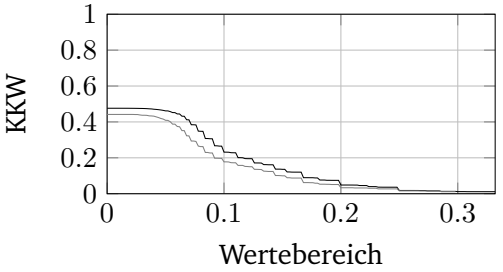
(a)realnamePropForefingerLeftDvorak



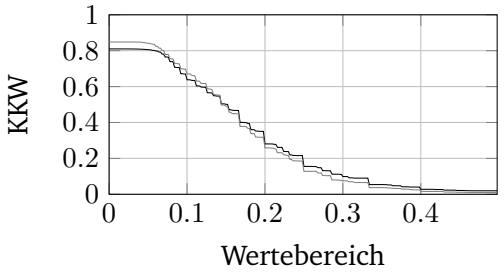
(b)realnamePropLittleFingerLeftDvorak



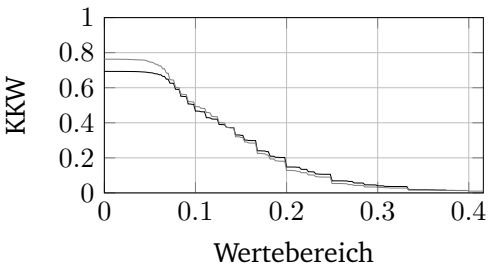
(c)realnamePropMiddleFingerLeftDvorak



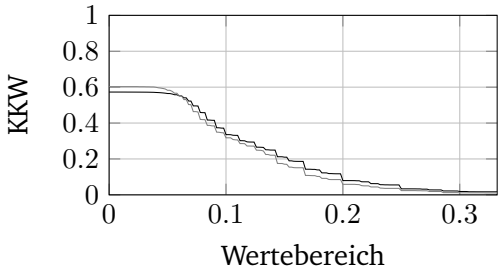
(d)realnamePropRingFingerLeftDvorak



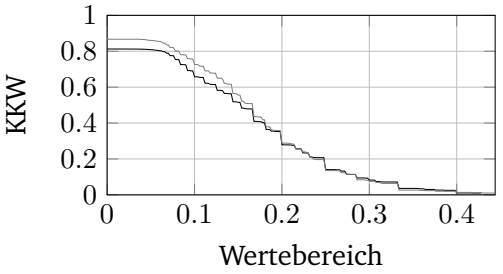
(e)realnamePropForefingerRightDvorak



(f)realnamePropLittleFingerRightDvorak

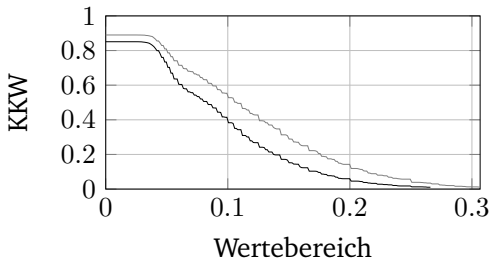


(g)realnamePropMiddleFingerRightDvorak

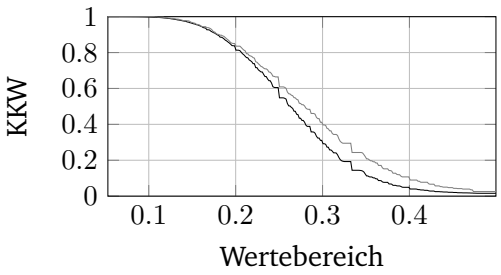


(h)realnamePropRingFingerRightDvorak

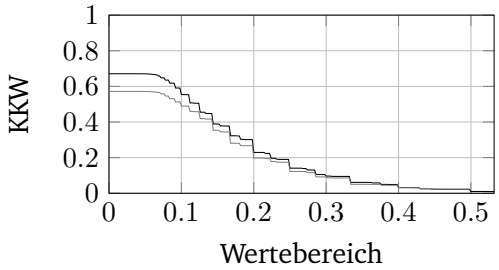
Abb. A.25.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ 15 / 16



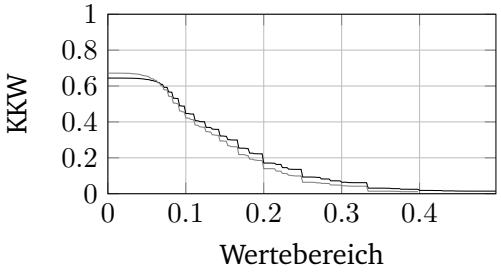
(a)emailPropSameFingerAsPrevQwertz



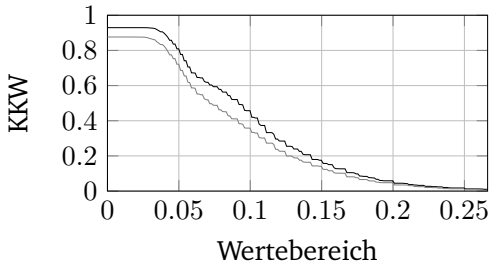
(b)homepagePropSameFingerAsPrevQwertz



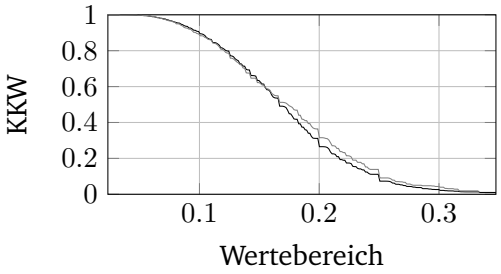
(c)namePropSameFingerAsPrevQwertz



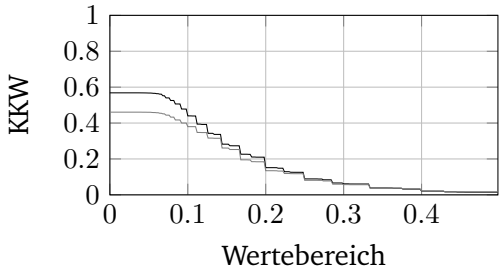
(d)realnamePropSameFingerAsPrevQwertz



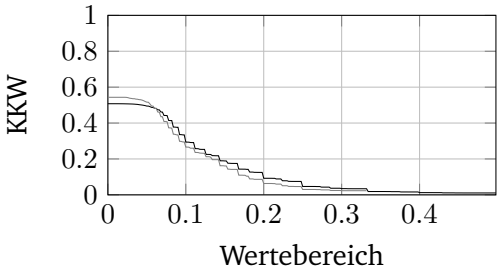
(e)emailPropSameFingerAsPrevDvorak



(f)homepagePropSameFingerAsPrevDvorak

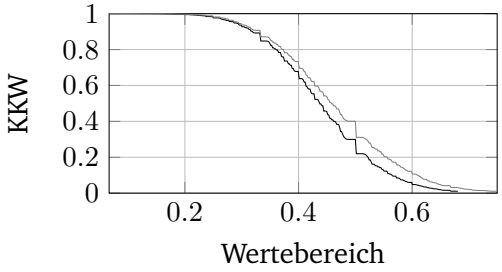


(g)namePropSameFingerAsPrevDvorak

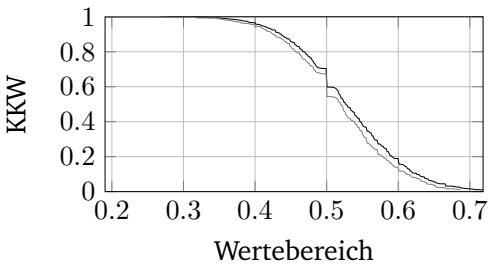


(h)realnamePropSameFingerAsPrevDvorak

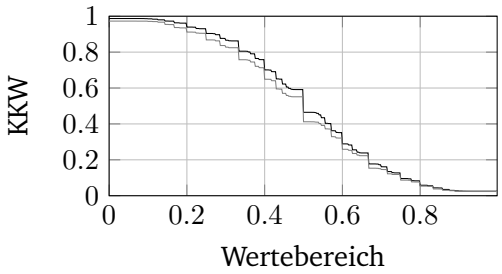
Abb. A.26.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propSameFingerAsPrev$



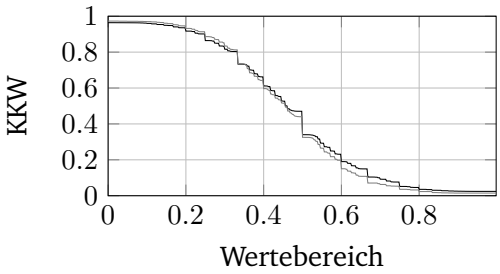
(a)emailPropSameHandAsPrevQwertz



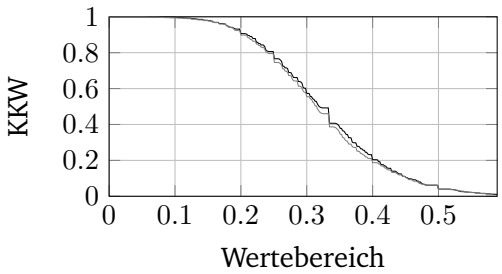
(b)homepagePropSameHandAsPrevQwertz



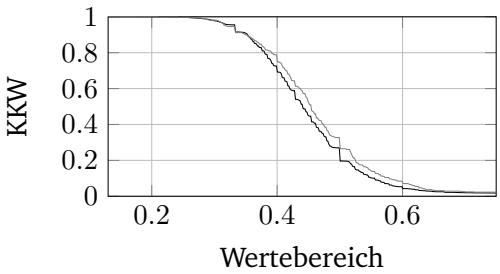
(c)namePropSameHandAsPrevQwertz



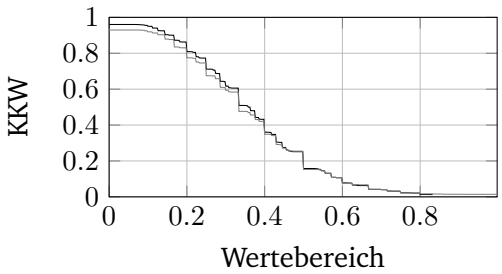
(d)realnamePropSameHandAsPrevQwertz



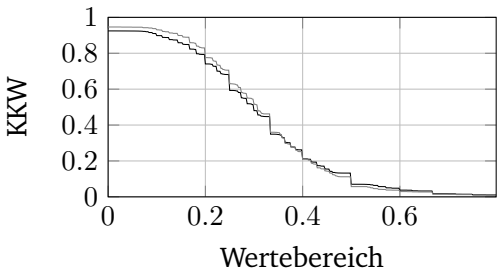
(e)emailPropSameHandAsPrevDvorak



(f)homepagePropSameHandAsPrevDvorak

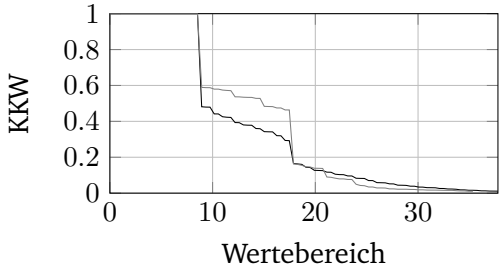


(g)namePropSameHandAsPrevDvorak

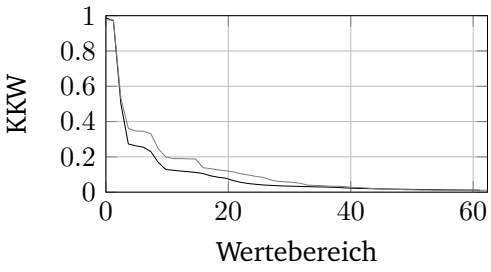


(h)realnamePropSameHandAsPrevDvorak

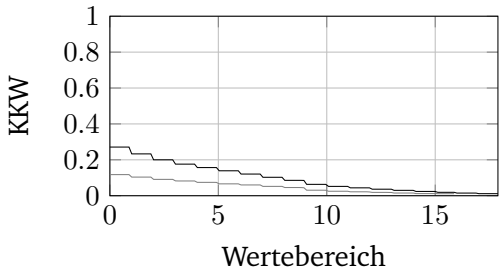
Abb. A.27.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propSameHandAsPrev$



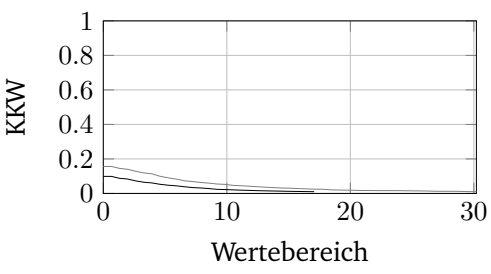
(a)emailDistanceOnKeyboardQwertz



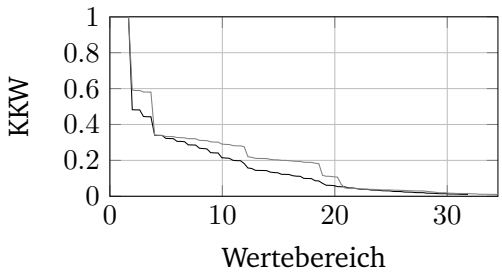
(b)homepageDistanceOnKeyboardQwertz



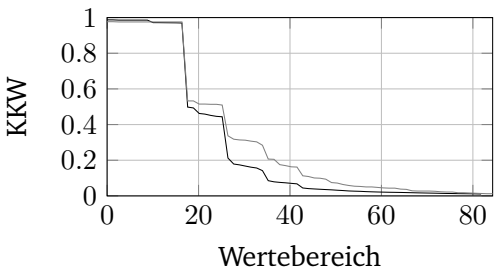
(c)nameDistanceOnKeyboardQwertz



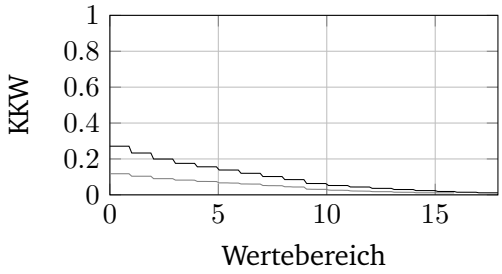
(d)realnameDistanceOnKeyboardQwertz



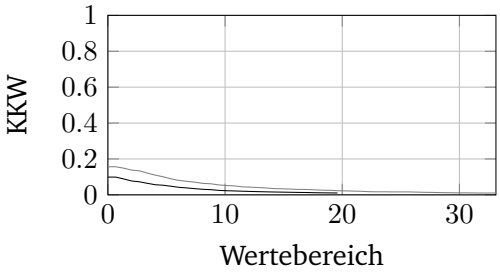
(e)emailDistanceOnKeyboardDvorak



(f)homepageDistanceOnKeyboardDvorak



(g)nameDistanceOnKeyboardDvorak



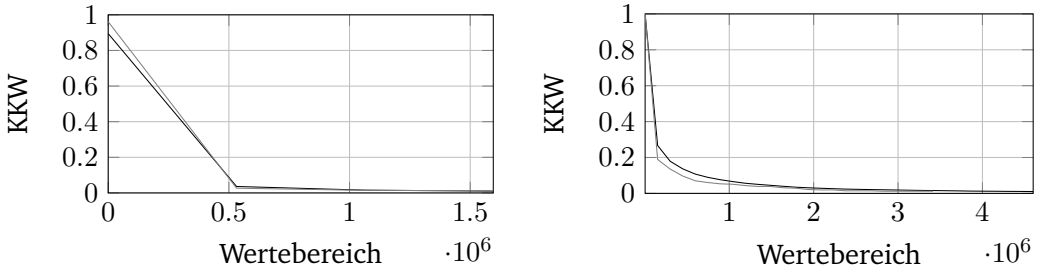
(h)realnameDistanceOnKeyboardDvorak

Abb. A.28.: Komplementäre Empirische Verteilungsfunktionen des Meta-Features $distanceOnKeyboard$

A.4.1 Auf Log-Daten basierende Features

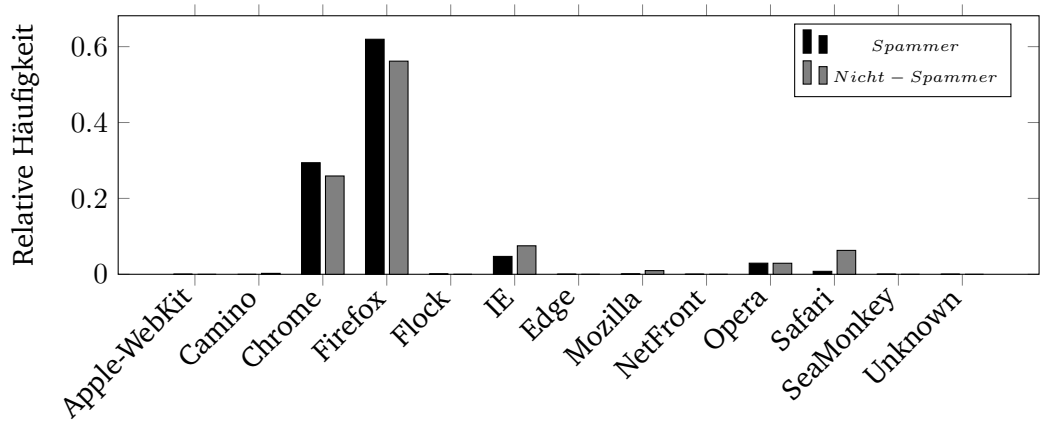
Tab. A.8.: Auf Log-Daten basierende Features

Feature Name	IG	Chi-Square
$activationDuration$	0,001263	18,93
$browser$	0,005391	186,64
os	0,011629	292,87
$refererFromBibsonomy$	0,000339	3,96
$registrationDuration$	0,002448	30,55

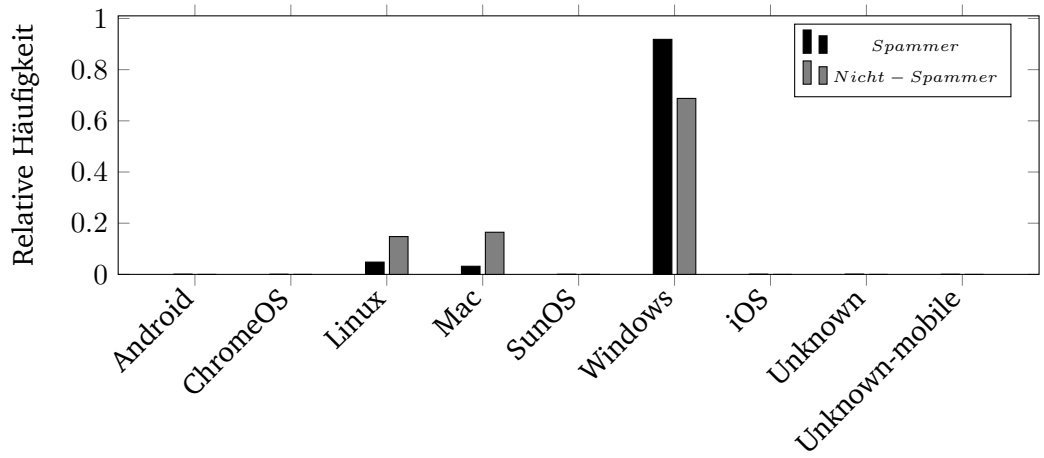


(a)registrationDuration **(b)**activationDuration

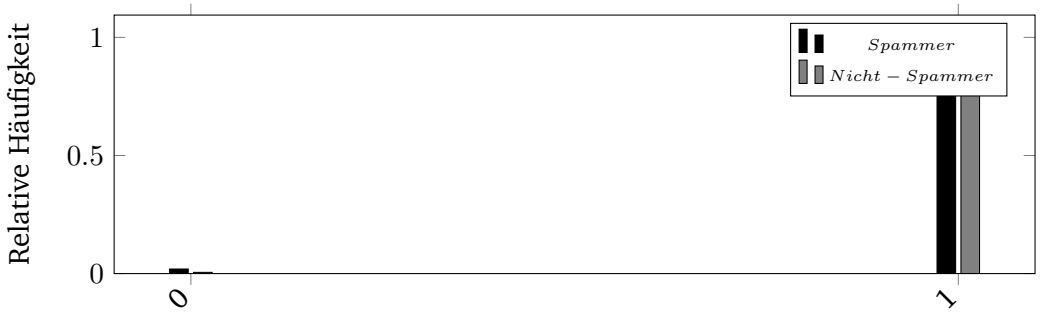
Abb. A.29.: Komplementäre Empirische Verteilungsfunktionen der Feature $registrationDuration$ & $activationDuration$



(a)browser



(b)os



(c)refererFromBibsonomy

Abb. A.30.: Werte-Verteilungen der Feature $browser$, os & $refererFromBibsonomy$

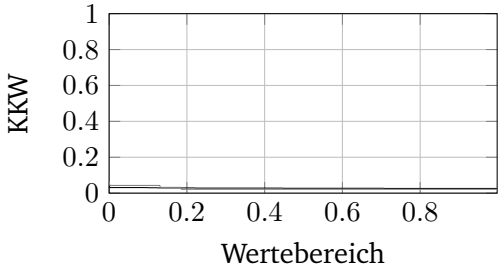
A.4.2 Interaktions-basierende Features

Tab. A.9.: Interaktions-basierendeFeatures 1 / 2

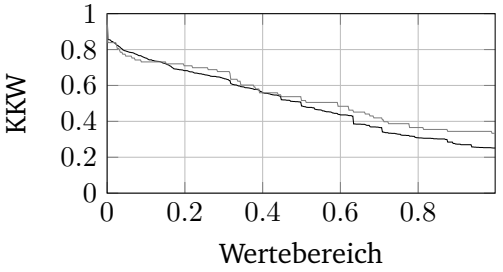
Feature Name	IG	Chi-Square
$maxAcceleration0$	0,000000	0,00
$maxAcceleration1$	0,000000	0,00
$maxAcceleration2$	0,000000	0,00
$maxAcceleration3$	0,000000	0,00
$maxAcceleration4$	0,000000	0,00
$maxAcceleration5$	0,000000	0,00
$maxAcceleration6$	0,000000	0,00
$maxAcceleration7$	0,000000	0,00
$maxClickTime$	0,000000	0,00
$maxDwellTime$	0,000000	0,00
$maxFlightTime$	0,000000	0,00
$maxPauseNClick$	0,000000	0,00
$maxVelocity0$	0,000000	0,00
$maxVelocity1$	0,000000	0,00
$maxVelocity2$	0,000000	0,00
$maxVelocity3$	0,000000	0,00
$maxVelocity4$	0,000000	0,00
$maxVelocity5$	0,000000	0,00
$maxVelocity6$	0,000000	0,00
$maxVelocity7$	0,000000	0,00
$meanAcceleration0$	0,000000	0,00
$meanAcceleration1$	0,000000	0,00
$meanAcceleration2$	0,000000	0,00
$meanAcceleration3$	0,000000	0,00
$meanAcceleration4$	0,000000	0,00
$meanAcceleration5$	0,000000	0,00
$meanAcceleration6$	0,000000	0,00
$meanAcceleration7$	0,000000	0,00
$meanClickTime$	0,000000	0,00
$meanDwellTime$	0,000000	0,00
$meanFlightTime$	0,000000	0,00
$meanPauseNClick$	0,000000	0,00

Tab. A.10.: Interaktions-basierendeFeatures 2 / 2

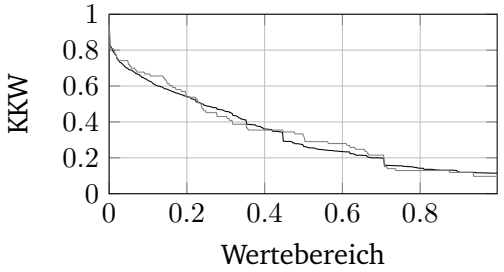
Feature Name	IG	Chi-Square
$meanVelocity0$	0,000000	0,00
$meanVelocity1$	0,000000	0,00
$meanVelocity2$	0,000000	0,00
$meanVelocity3$	0,000000	0,00
$meanVelocity4$	0,000000	0,00
$meanVelocity5$	0,000000	0,00
$meanVelocity6$	0,000000	0,00
$meanVelocity7$	0,000000	0,00
$minAcceleration0$	0,000000	0,00
$minAcceleration1$	0,000000	0,00
$minAcceleration2$	0,000000	0,00
$minAcceleration3$	0,000000	0,00
$minAcceleration4$	0,000000	0,00
$minAcceleration5$	0,000000	0,00
$minAcceleration6$	0,000000	0,00
$minAcceleration7$	0,000000	0,00
$minClickTime$	0,000000	0,00
$minDwellTime$	0,000000	0,00
$minFlightTime$	0,000000	0,00
$minPauseNClick$	0,000000	0,00
$minVelocity0$	0,000000	0,00
$minVelocity1$	0,000000	0,00
$minVelocity2$	0,000000	0,00
$minVelocity3$	0,000000	0,00
$minVelocity4$	0,000000	0,00
$minVelocity5$	0,000000	0,00
$minVelocity6$	0,000000	0,00
$minVelocity7$	0,000000	0,00
$movedDistance0$	0,000000	0,00
$movedDistance1$	0,000000	0,00
$movedDistance2$	0,000000	0,00
$movedDistance3$	0,000000	0,00
$movedDistance4$	0,000000	0,00
$movedDistance5$	0,000000	0,00
$movedDistance6$	0,000000	0,00
$movedDistance7$	0,000000	0,00



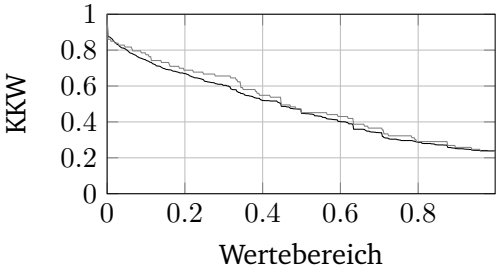
(a)minAcceleration0



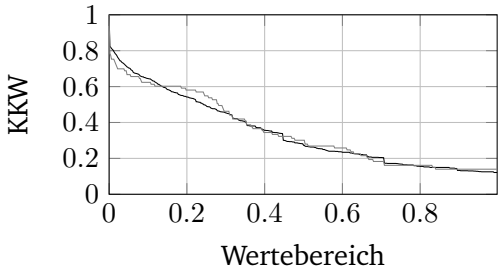
(b)minAcceleration1



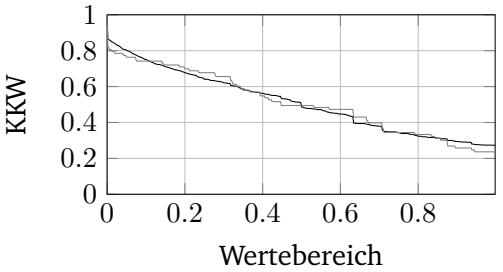
(c)minAcceleration2



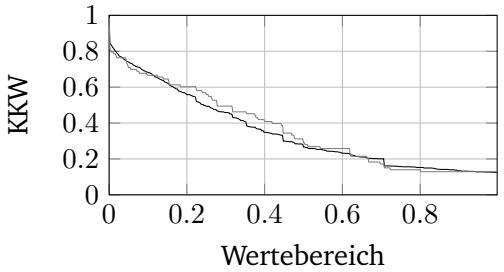
(d)minAcceleration3



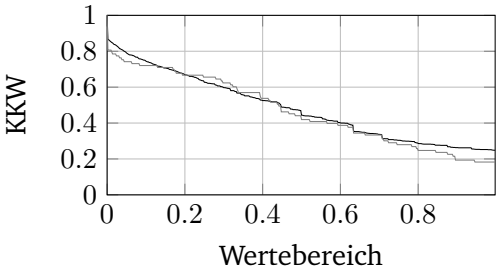
(e)minAcceleration4



(f)minAcceleration5

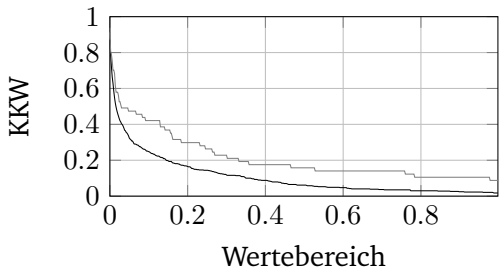


(g)minAcceleration6

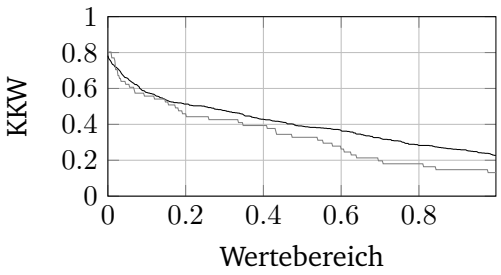


(h)minAcceleration7

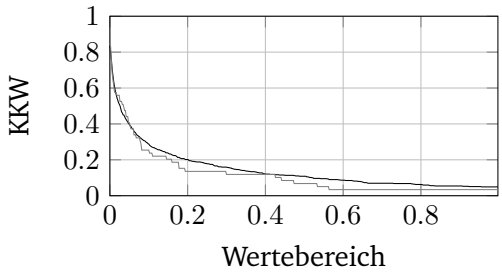
Abb. A.31.: Komplementäre Empirische Verteilungsfunktionen des Meta-Feature *acceleration*



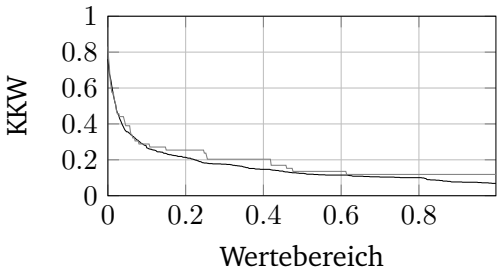
(a)meanAcceleration0



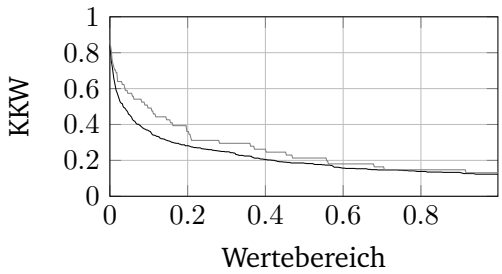
(b)meanAcceleration1



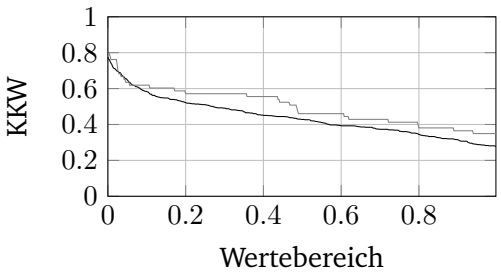
(c)meanAcceleration2



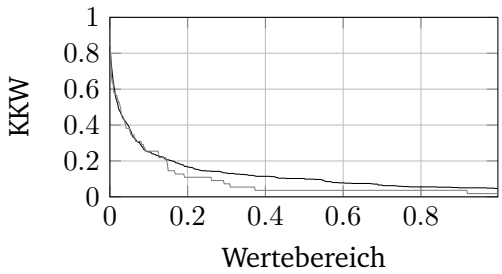
(d)meanAcceleration3



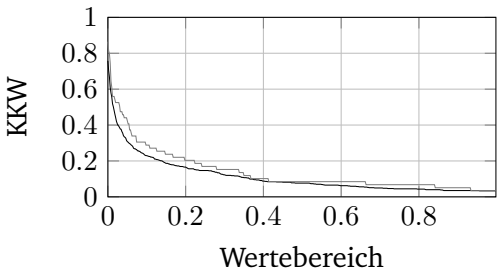
(e)meanAcceleration4



(f)meanAcceleration5

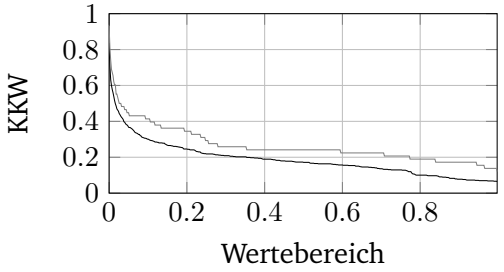


(g)meanAcceleration6

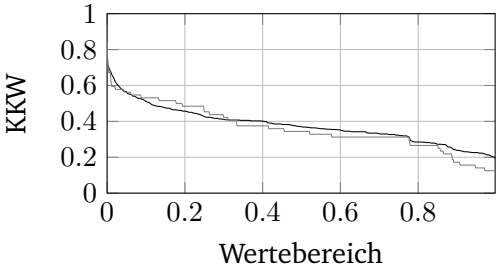


(h)meanAcceleration7

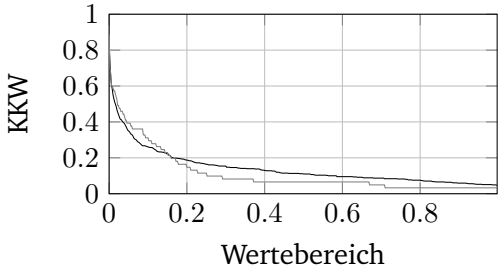
Abb. A.32.: Komplementäre Empirische Verteilungsfunktionen des Meta-Feature *acceleration*



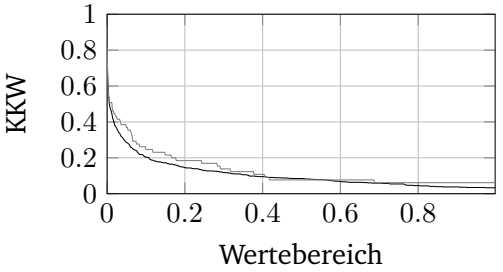
(a)maxAcceleration0



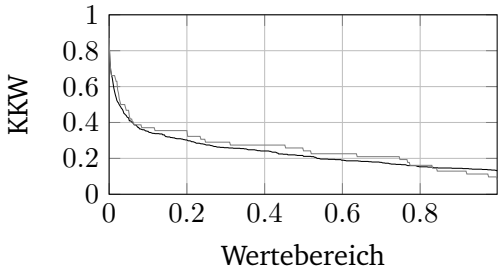
(b)maxAcceleration1



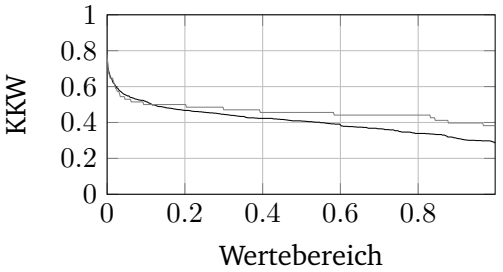
(c)maxAcceleration2



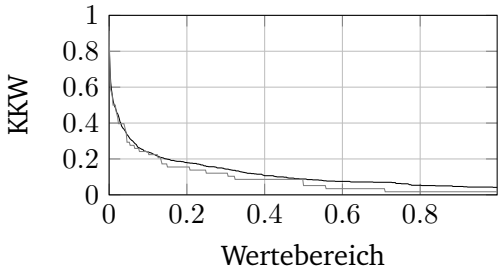
(d)maxAcceleration3



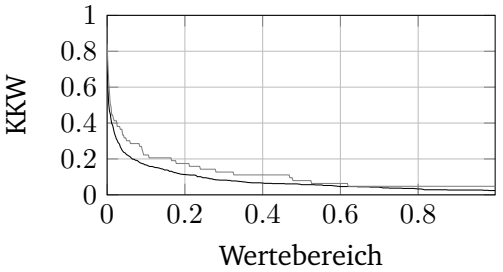
(e)maxAcceleration4



(f)maxAcceleration5

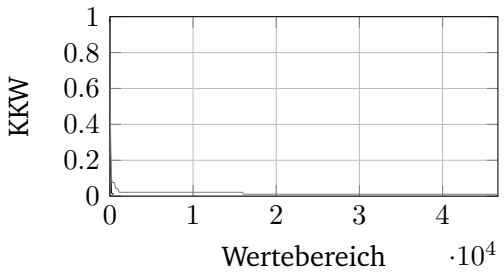


(g)maxAcceleration6

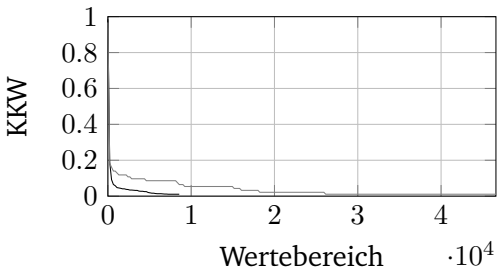


(h)maxAcceleration7

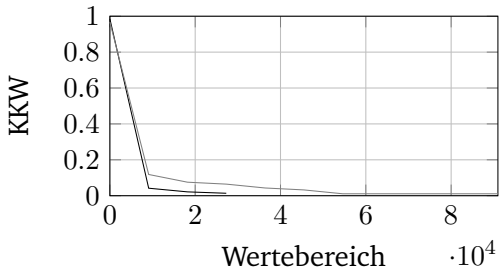
Abb. A.33.: Komplementäre Empirische Verteilungsfunktionen des Meta-Feature *acceleration*



(a)minClickTime

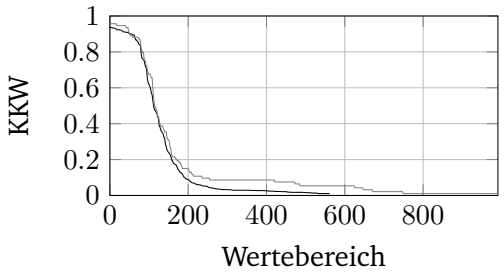


(b)meanClickTime

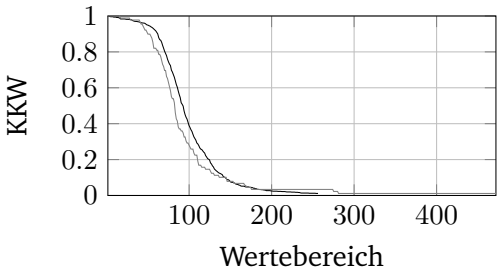


(c)maxClickTime

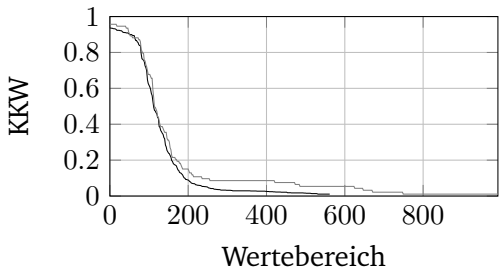
Abb. A.34.: Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $clickTime$



(a)minDwellTime

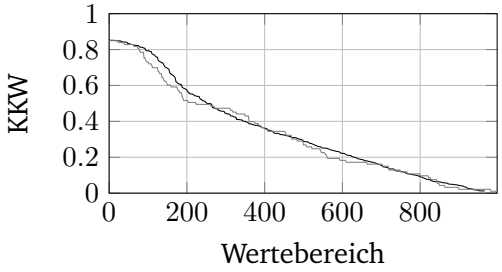


(b)meanDwellTime

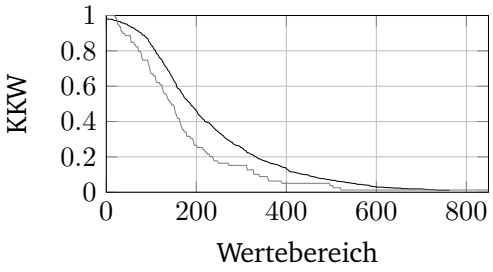


(c)maxDwellTime

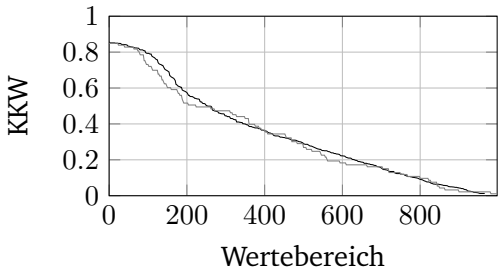
Abb. A.35.: Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $dwellTime$



(a)minFlightTime

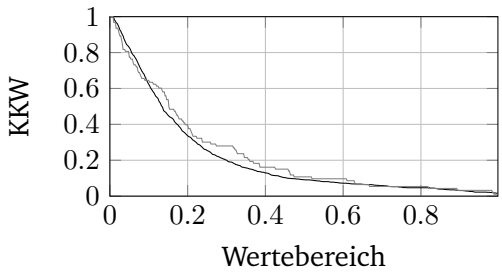


(b)meanFlightTime

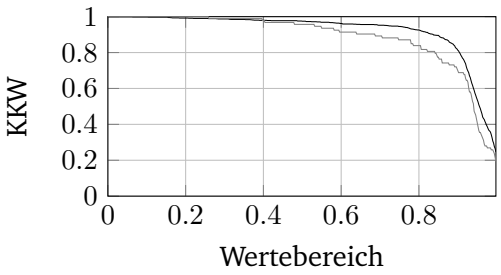


(c)maxFlightTime

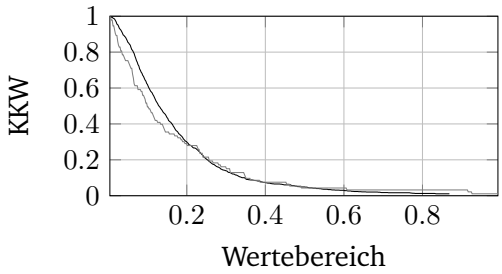
Abb. A.36.: Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $flightTime$



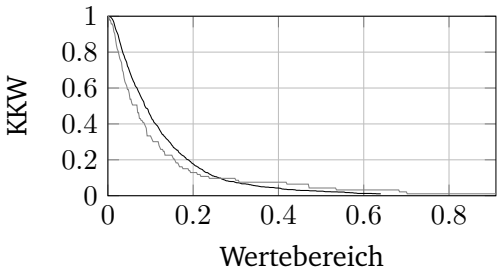
(a)movedDistance0



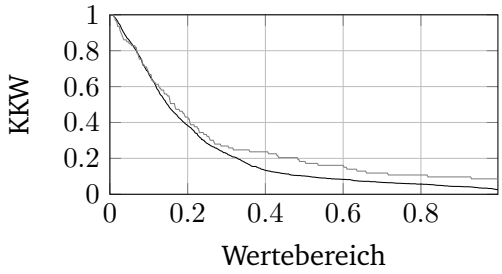
(b)movedDistance1



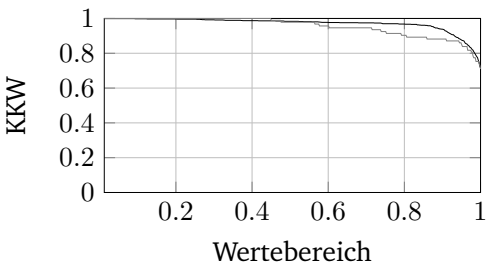
(c)movedDistance2



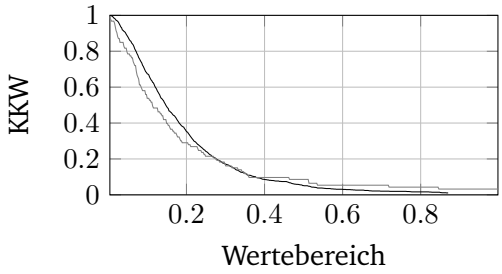
(d)movedDistance3



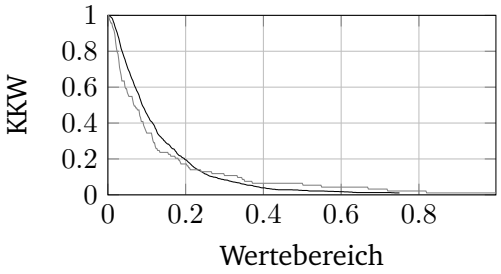
(e)movedDistance4



(f)movedDistance5

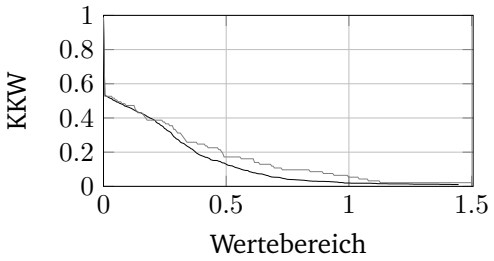


(g)movedDistance6

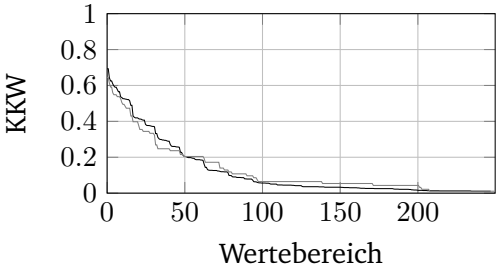


(h)movedDistance7

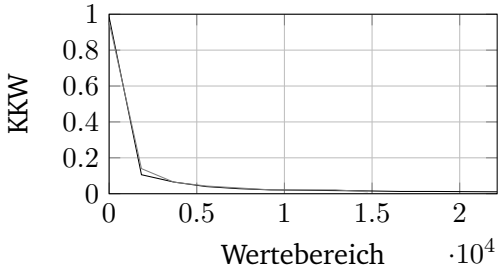
Abb. A.37.: Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $movedDistance$



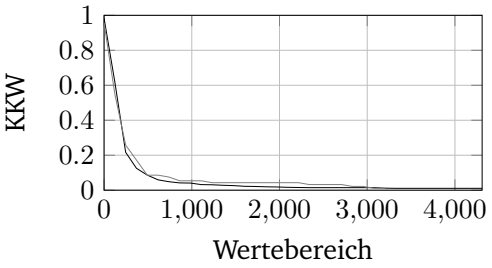
(a)mouseMoveTime



(b)minPauseNClick

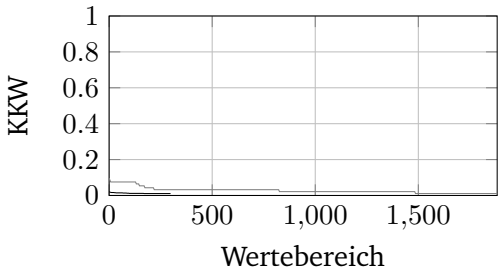


(c)maxPauseNClick

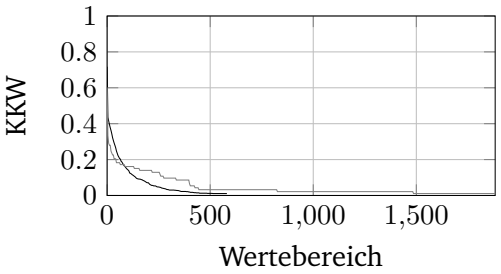


(d)meanPauseNClick

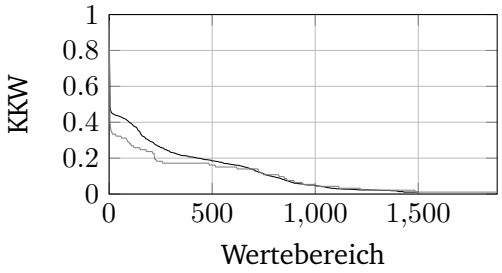
Abb. A.38.: Komplementäre Empirische Verteilungsfunktionen des Feature $mouseMoveTime$ & Meta-Feature $pauseNClick$



(a)minTraveledDist

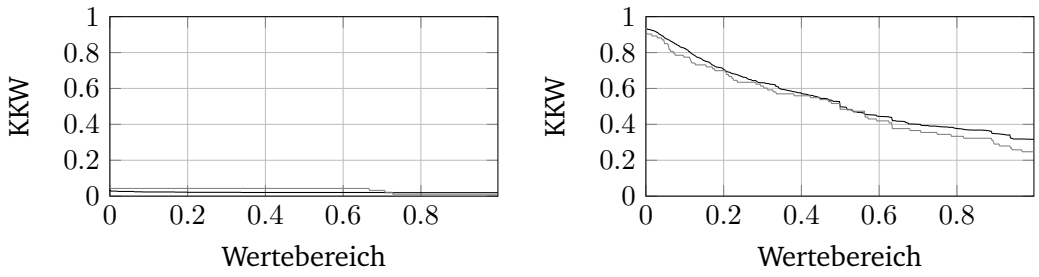


(b)meanTraveledDist



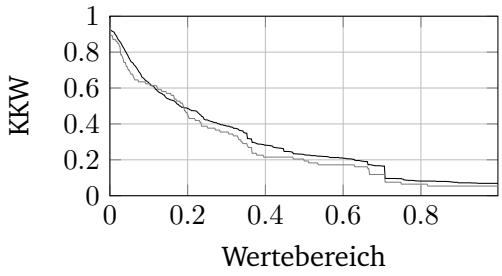
(c)maxTraveledDist

Abb. A.39.: Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $traveledDist$

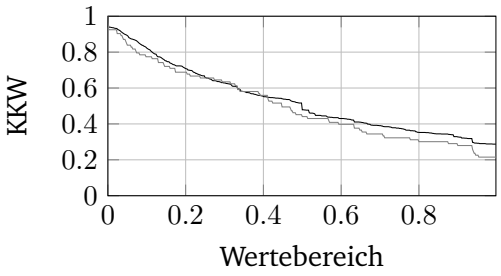


(a)minVelocity0

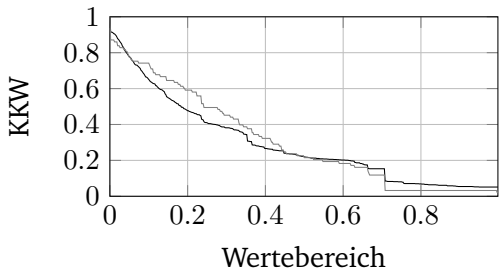
(b)minVelocity1



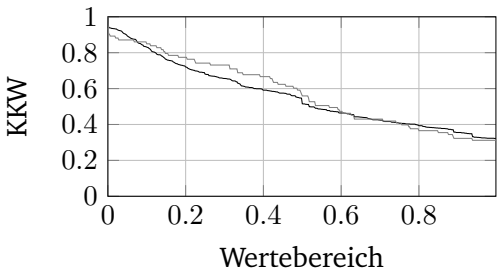
(c)minVelocity2



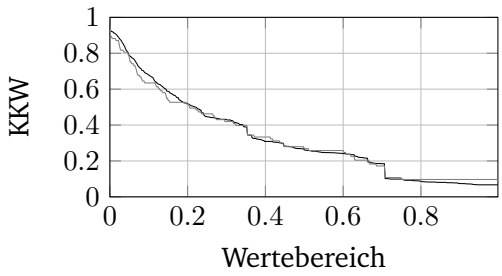
(d)minVelocity3



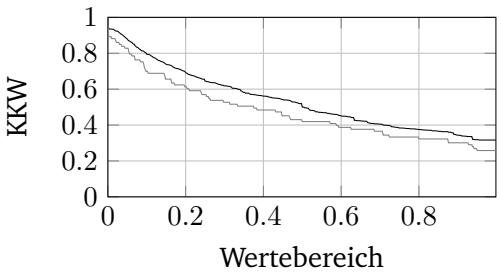
(e)minVelocity4



(f)minVelocity5

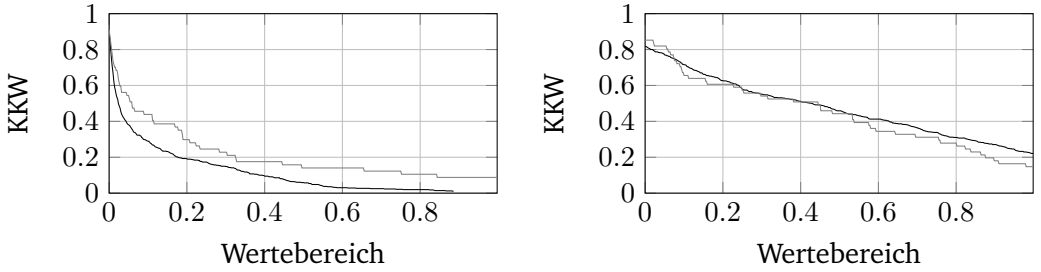


(g)minVelocity6

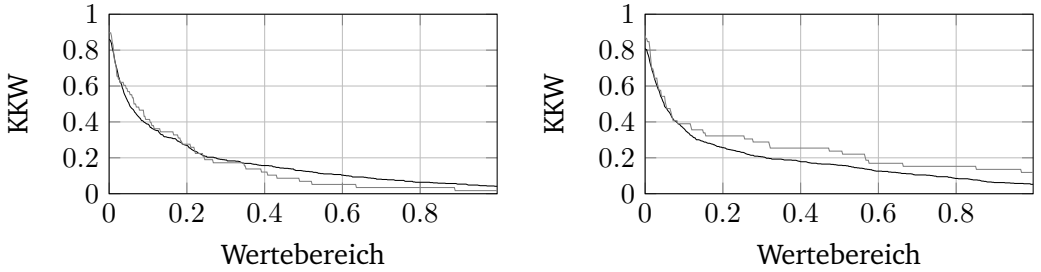


(h)minVelocity7

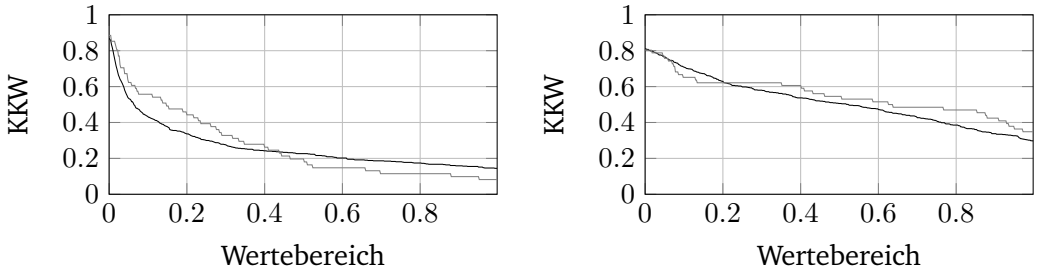
Abb. A.40.: Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $velocity$



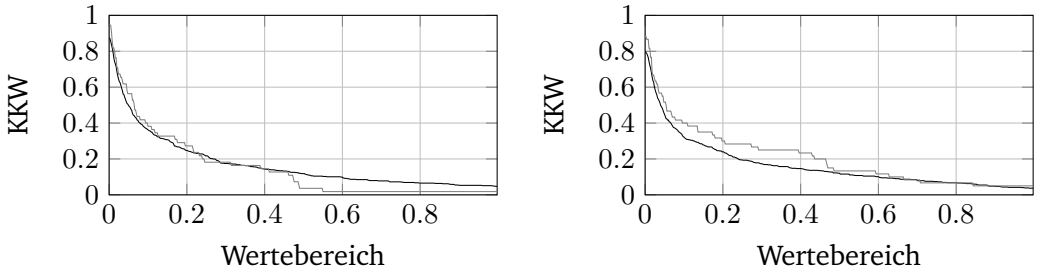
(a)meanVelocity0 **(b)**meanVelocity1



(c)meanVelocity2 **(d)**meanVelocity3

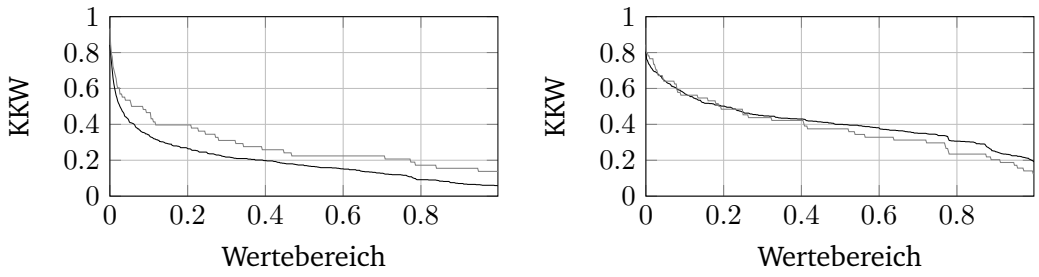


(e)meanVelocity4 **(f)**meanVelocity5



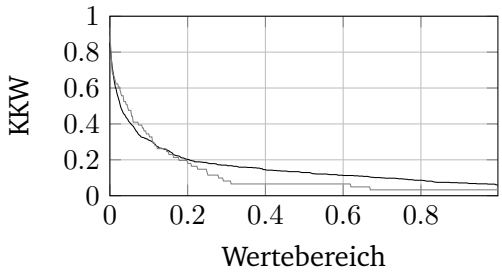
(g)meanVelocity6 **(h)**meanVelocity7

Abb. A.41.: Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $velocity$

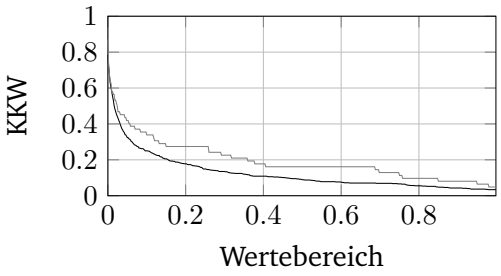


(a)maxVelocity0

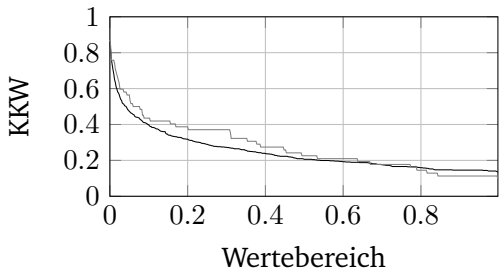
(b)maxVelocity1



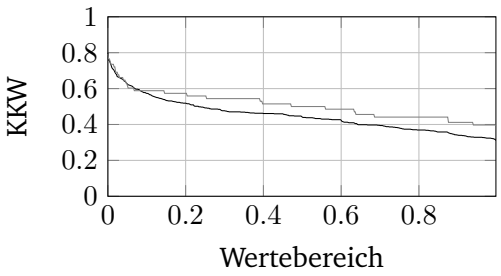
(c)maxVelocity2



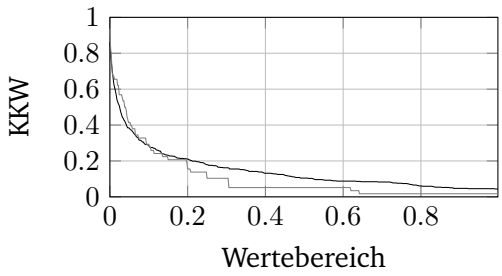
(d)maxVelocity3



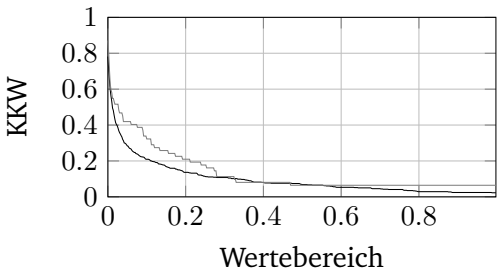
(e)maxVelocity4



(f)maxVelocity5



(g)maxVelocity6



(h)maxVelocity7

Abb. A.42.: Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $velocity$

Literatur

[1]Md Liakat Ali, Charles C. Tappert, Meikang Qiu und John V. Monaco. „Authentication and Identification Methods Used in Keystroke Biometric Systems.“ In: *HPCC/CSS/ICESS*. IEEE, 2015, S. 1424–1429 (zitiert auf Seite 12).

[2]Martin Atzmueller, Florian Lemmerich, Beate Krause und Andreas Hotho. „Who are the Spammers? Understandable Local Patterns for Concept Description“. In: *7th Conference on Computer Methods and Systems*. ISBN 83-916420-5-4. Krakow, Poland, 2009 (zitiert auf Seite 11).

[3]F. Benevenuto, G. Magno, T. Rodrigues und V. Almeida. „Detecting Spammers on Twitter“. In: *Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*. Redmond, USA, Juli 2010 (zitiert auf Seite 45).

[4]E. Bizzi, N. Accornero, W. Chapple und N. Hogan. „Posture control and trajectory formation during arm movment“. In: *The Journal of Neuroscience* 4.11 (1984), S. 2738–2744 (zitiert auf Seite 41).

[5]Enrico Blanzieri und Anton Bryl. „A survey of learning-based techniques of email spam filtering“. In: *Artif. Intell. Rev.* 29 (1 2008), S. 63–92 (zitiert auf Seite 11).

[6]J. Bonneau, C. Herley, P. C. van Oorschot und F. Stajano. „Passwords and the Evolution of Imperfect Authentication“. In: *Comm. ACM* 58.7 (2015), S. 78–87 (zitiert auf Seite 12).

[7]Beate Navarro Bullock, Hana Lerch, Alexander Rossnagel, Andreas Hotho und Gerd Stumme. „Privacy-aware spam detection in social bookmarking systems“. In: *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*. i-KNOW ’11. Graz, Austria: ACM, 2011, 15:1–15:8 (zitiert auf Seite 11).

[8]Carlos Castillo, Debora Donato, Luca Becchetti et al. „A reference collection for web spam“. In: *SIGIR Forum* 40 (2 2006), S. 11–24 (zitiert auf Seite 3).

[9]Anita Crescenzi, Diane Kelly und Leif Azzopardi. „Impacts of Time Constraints and System Delays on User Experience.“ In: *CHIIR*. Hrsg. von Diane Kelly, Robert Capra, Nicholas J. Belkin, Jaime Teevan und Pertti Vakkari. ACM, 2016, S. 141–150 (zitiert auf Seite 59).

[10]Harris Drucker, Vladimir Vapnik und Dongui Wu. „Support Vector Machines for Spam Categorization“. In: *IEEE Transactions on Neural Networks* 10.5 (1999), S. 1048–1054 (zitiert auf Seite 45).

[11]Tom Fawcett. „An introduction to ROC analysis“. In: *Pattern Recognition Letters* 27.8 (2006), S. 861–874 (zitiert auf den Seiten 6–9).

[12]Clint Feher, Yuval Elovici, Robert Moskovitch, Lior Rokach und Alon Schclar. „User identity verification via mouse dynamics.“ In: *Inf. Sci.* 201 (2012), S. 19–36 (zitiert auf den Seiten 12, 41, 42).

[13]Zoltán Gyöngyi und Hector Garcia-Molina. „Web Spam Taxonomy“. In: *AIRWeb*. 2005, S. 39–47 (zitiert auf Seite 3).

[14]Jürgen Hedderich und Lothar Sachs. *Angewandte Statistik : Methodensammlung mit R*. Berlin ; Heidelberg: Springer, 2012 (zitiert auf den Seiten 9, 10).

[15]Xia Hu, Jiliang Tang, Yanchao Zhang und Huan Liu. „Social Spammer Detection in Microblogging.“ In: *IJCAI*. Hrsg. von Francesca Rossi. IJCAI/AAAI, 2013 (zitiert auf Seite 12).

[16]*IBM X-Force Threat Intelligence Quarterly, 2Q 2015*. Techn. Ber. IBM Corporation, Juni 2015 (zitiert auf Seite 30).

[17]Nathalie Japkowicz und Shaju Stephen. „The Class Imbalance Problem: A Systematic Study.“ In: *Intelligent Data Analysis* 6.5 (2002), S. 429–449 (zitiert auf Seite 46).

[18]Iftikhar Ahmed Khan, Willem-Paul Brinkman, Nick Fine und Robert M. Hierons. „Measuring personality from keyboard and mouse use.“ In: *ECCE*. Hrsg. von Julio Abascal, Inmaculada Fajardo und Ian Oakley. ACM, 2008, S. 38 (zitiert auf den Seiten 12, 41).

[19]Kenji Kira und Larry A. Rendell. „A Practical Approach to Feature Selection.“ In: *ML*. Hrsg. von Derek H. Sleeman und Peter Edwards. Morgan Kaufmann, 4. Dez. 2002, S. 249–256 (zitiert auf Seite 45).

[20]Ron Kohavi. „A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection“. In: *IJCAI*. 1995, S. 1137–1145 (zitiert auf Seite 46).

[21]B. Krause, C. Schmitz, A. Hotho und G. Stumme. „The Anti-Social Tagger – Detecting Spam in Social Bookmarking Systems“. In: *Proc. of the Fourth International Workshop on Adversarial Information Retrieval on the Web*. 2008 (zitiert auf den Seiten 1, 11, 13, 23, 26, 28, 45, 46, 63).

[22]David M. W. Powers. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*. Techn. Ber. SIE-07-001. Adelaide, Australia: School of Informatics und Engineering, Flinders University, 2007 (zitiert auf Seite 7).

[23]Alireza Saberi, Mojtaba Vahidi und Behrouz Minaei-Bidgoli. „Learn to Detect Phishing Scams Using Learning and Ensemble ?Methods.“ In: *Web Intelligence/IAT Workshops*. IEEE Computer Society, 2007, S. 311–314 (zitiert auf Seite 45).

[24]D. Shanmugapriya und G. Padmavathi. „A Survey of Biometric keystroke Dynamics: Approaches, Security and Challenges“. In: *CoRR* abs/0910.0817 (2009) (zitiert auf Seite 42).

[25]C. E. Shannon. „A Mathematical Theory of Communication“. In: *Bell Systems Technical Journal* 27 (1948), S. 623–656 (zitiert auf Seite 25).

[26]Mugdha Sharma und Jasmeen Kaur. „A Novel Data Mining Approach for Detecting Spam Emails using Robust Chi-Square Features.“ In: *WCI*. Hrsg. von Indu Nair, Sushmita Mitra, Ljiljana Trajkovic et al. ACM, 2015, S. 49–53 (zitiert auf Seite 45).

[27]Fergus Toolan und Joe Carthy. „Feature selection for Spam and Phishing detection“. In: *2010 eCrime Researchers Summit*. Institute of Electrical & Electronics Engineers (IEEE), 2010 (zitiert auf Seite 46).

[30]Y. Yang und J. Petersen. „A comparative study on feature selection in text categorization“. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. 1997 (zitiert auf den Seiten 45, 46).

[31]R. Zafarani und H. Liu. „10 Bits of Surprise: Detecting Malicious Users with Minimum Information.“ In: *Proceedings CIKM*. ACM, 2015, S. 423–431 (zitiert auf den Seiten 11, 23–25, 28, 34, 45, 63).

[32]Nan Zheng, Aaron Paloski und Haining Wang. „An efficient user verification system via mouse movements.“ In: *ACM Conference on Computer and Communications Security*. Hrsg. von Yan Chen, George Danezis und Vitaly Shmatikov. ACM, 2011, S. 139–150 (zitiert auf Seite 12).

[33]Yin Zhu, Xiao Wang, ErHeng Zhong et al. „Discovering Spammers in Social Networks.“ In: *AAAI*. Hrsg. von Jörg Hoffmann 0001 und Bart Selman. AAAI Press, 2012 (zitiert auf Seite 12).

Webseiten

[@28]Wikipedia, Die freie Enzyklopädie. *Support Vector Machine*. 2016. URL: `https://de.wikipedia.org/wiki/Support_Vector_Machine` (besucht am 10. Mai 2016) (zitiert auf Seite 5).

[@29]Wikipedia, Die freie Enzyklopädie. *Zehnfingersystem*. 2016. URL: `https://de.wikipedia.org/wiki/Zehnfingersystem` (besucht am 10. Mai 2016) (zitiert auf Seite 34).

Abbildungsverzeichnis

2.1 Beispiel SVM mit 2-dim. Daten und linearer Kernelfunktion. Quelle: [@28] 5

4.1 Mausbewegung und Mausklicks bei der Registrierung von Bibsonomy . 15
4.2 Automat zur Entscheidung einer gültigen Registrierung 17

5.1 Klassendiagramm Features 21
5.2 Empirische Verteilungsfunktionen im Meta-Features $length$ 22
5.3 Empirische Verteilungsfunktionen im Meta-Feature $numUniqueAlphLetters$ 23
5.4 Empirische Verteilungsfunktionen im Meta-Feature $numMaxTimesLetterRep$ 24
5.5 Empirische Verteilungsfunktionen im Meta-Feature $entropy$ 25
5.6 Empirische Verteilungsfunktionen im Meta-Features $infoSupr$ 26
5.7 Werte-Verteilungen im Meta-Feature $digit$ 27
5.8 Empirische Verteilungsfunktionen im Meta-Features $numDigit$ & $propDigit$ 27
5.9 CCFD-Verteilung des Features $realnameParts$ 28
5.10 Werte-Verteilungen der Feature $nameEqualMail$ & $nameEqualMailEqualRealnname$ 29
5.11 Werte-Verteilungen des Features $uniMail$ 29
5.12 Top-5 Ursprungsländer der Spammer vereint mit Top-5 Ursprungsländer der Nicht-Spammer 30
5.13 Werte-Verteilungen der Feature $regDay$ & $regHour$ 31
5.14 Empirische Verteilungsfunktionender Features $lastnameCount$ & $spamIP$ im Meta-Feature $count$ 32
5.15 Empirische Verteilungsfunktionen der Features $lastnameRatio$ & $tldRatio$ im Meta-Feature $ratio$ 33
5.16 Zuordnung der Tasten im Zehfingersystem. Quelle: [@29] 34
5.17 Empirische Verteilungsfunktionen im Meta-Feature $propRow$ 35
5.18 Empirische Verteilungsfunktionen des Meta-Feature $propFinger$ . . . 36
5.19 Empirische Verteilungsfunktionen des Meta-Feature $propSameFingerAsPrev$ 37
5.20 Empirische Verteilungsfunktionen des Meta-Feature $distanceOnKeyboard$ 38

5.21 Empirische Verteilungsfunktionen der Feature $registrationDuration$ & $activationDuration$ . . . 39
5.22 Werte-Verteilungen der Feature $browser$, os . . . 40
5.23 Empirische Verteilungsfunktionen der Feature $meanAcceleration7$, $meanDwellTime$ & $movedDistance0$ . . . 43

7.1 ROC für alle Feature der Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basierte Feature . . . 52
7.2 ROC je Kategorie . . . 53
7.4 ROC der Kategorien auf Log-Dateien basierte und Interaktions-basierte Feature . . . 54
7.6 ROC der 25 besten Feature nach IG und ChSq . . . 56

8.1 Klassendiagramm Anbindung an Bibsonomy . . . 60

A.1 Werte-Verteilungen des Meta-Features $digit$ . . . 67
A.2 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $entropy$ . . . 67
A.3 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $infoSupr$ . . . 68
A.4 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $length$ . . . 68
A.5 Werte-Verteilungen der Feature $a : nameEqualMail$ & $nameEqualMailEqualRealnname$ . . . 69
A.6 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $numDigit$ . . . 69
A.7 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $numUniqueAlphLetters$ . . . 70
A.8 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $numMaxTimesLetterRep$ . . . 70
A.9 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propDigit$ . . . 71
A.10 Komplementäre Empirische Verteilungsfunktionen des Feature $realnameParts$ . . . 71
A.11 Werte-Verteilungen der Feature $regDay$, $regHour$ & $uniMail$ . . . 72
A.12 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $count$ . . . 74
A.13 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $ratio$ 1 / 2 . . . 75
A.14 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$ . . . 79
A.15 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$ . . . 80

A.16 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$ . . . 81
A.17 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propRow$ . . . 82
A.18 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ . . . 83
A.19 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ . . . 84
A.20 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ . . . 85
A.21 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ . . . 86
A.22 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ . . . 87
A.23 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ . . . 88
A.24 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ . . . 89
A.25 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propFinger$ 15 / 16 . . . 90
A.26 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propSameFingerAsPrev$ . . . 91
A.27 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $propSameHandAsPrev$ . . . 92
A.28 Komplementäre Empirische Verteilungsfunktionen des Meta-Features $distanceOnKeyboard$ . . . 93
A.29 Komplementäre Empirische Verteilungsfunktionen der Feature $registrationDuration$ & $activationDuration$ . . . 94
A.30 Werte-Verteilungen der Feature $browser$, os & $referer FromBibsonomy$ . . . 95
A.31 Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $acceleration$ . . . 98
A.32 Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $acceleration$ . . . 99
A.33 Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $acceleration$ . . . 100
A.34 Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $clickTime$ . . . 101
A.35 Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $dwellTime$ . . . 101
A.36 Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $flightTime$ . . . 102

A.37 Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $movedDistance$ . . . 103
A.38 Komplementäre Empirische Verteilungsfunktionen des Feature $mouseMoveTime$ & Meta-Feature $pauseNClick$ . . . 104
A.39 Komplementäre Empirische Verteilungsfunktionen des Feature Meta-Feature $traveledDist$ . . . 104
A.40 Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $velocity$ . . . 105
A.41 Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $velocity$ . . . 106
A.42 Komplementäre Empirische Verteilungsfunktionen des Meta-Feature $velocity$ . . . 107

Tabellenverzeichnis

2.1 Wahrheitsmatrix eines Klassifikators 6

4.1 Benutzer-Statistik Bibsonomy-Datenbank 14
4.2 Benutzer-Statistik: Event-Log der Registrierung 15
4.3 Gespeicherte Daten im Log nach Event-Typ 16
4.4 Benutzer-Statistik: Apache-Log 19

6.1 Beispiel-Daten: Feature-Daten wurden aus allen Daten erzeugt 47
6.2 Beispiel-Daten: Feature-Daten wurden aus einem Teil der Daten erzeugt 47

7.1 Evaluations-Ergebnisse für alle Feature der Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basierte Feature 51
7.2 Evaluations-Ergebnisse nach Kategorien 53
7.3 Evaluations-Ergebnisse der Kosten-Sensitiven-Klassifikatoren 55
7.4 Evaluations-Ergebnisse der Feature Selection 56
7.5 Die 25 besten Features absteigend sortiert nach ihrem Informationsgewinn 57
7.6 Die 25 besten Features aus den Kategorien Sprach-, Umwelt-, Tastatur- und Populations-basiert absteigend sortiert nach ihrem Informationsgewinn 58

8.1 Benchmark REST-Server 62

A.1 Sprach-basierte Features 1 / 2 65
A.2 Sprach-basierte Features 2 / 2 66
A.3 Umwelt-basierte Features 72
A.4 Populations-basierte Features 73
A.5 Tastatur-basierte Features 1 / 3 76
A.6 Tastatur-basierte Features 2 / 3 77
A.7 Tastatur-basierte Features 3 / 3 78
A.8 Auf Log-Daten basierende Features 94
A.9 InteraktionsbasierendeFeatures 1 / 2 96
A.10 InteraktionsbasierendeFeatures 2 / 2 97