Köln, den 27. September 2017

Universität zu Köln
Sprachliche Informationsverarbeitung
Themensteller: Dr. Jürgen Hermes

Machine-Learning-Verfahren zur Klassifikation von Stellenausschreibungen

vorgelegt von

Johanna Binnewitt
Matrikelnummer 5429692
E-Mail: jbinnewitt@gmail.com
Dellbrücker Hauptstr. 89
51069 Köln

Inhaltsverzeichnis

1. Einleitung

In der heutigen Zeit spielt sich ein großer Teil der Kommunikation im Internet ab. Neben sozialen Netzwerken und informativen Webseiten tummelt sich eine breite Masse an Portalen für Stellenausschreibungen, Immobilien oder Dienstleistungen im World Wide Web. Da diese Daten in elektronischer Form vorliegen, ist es möglich, sie maschinell zu verarbeiten, um so ihre Inhalte besser zu erfassen. Dies kann hilfreich sein, da die Masse der Nachrichten oder Anzeigen oft so schnell wächst, dass ein Mensch Schwierigkeiten hat, sie zu überblicken. Viele dieser Daten liegen als natursprachlicher Text vor, deshalb können Methoden aus der Computerlinguistik dabei helfen, diese Daten zu strukturieren. Dies ist auch der Fall bei der Klassifikation von Stellenausschreibungen. In der vorliegenden Arbeit geht es deshalb darum, Stellenanzeigen maschinell zu verarbeiten, sodass sie in vorgefertigte Kategorien sortiert werden können. Dazu werden Methoden des maschinellen Lernens aus dem Bereich der Textklassifikation verwendet, die dabei helfen sollen, eine Menge von Stellenausschreibungen für den Menschen oder für weitere Verarbeitungsmechanismen erfassbarer zu machen.

Im Folgenden werden zunächst die genaue Fragestellung und die damit verbundenen Daten vorgestellt. Anschließend wird in Kapitel 3 auf die Problematik der Klassifikation eingegangen und erläutert, welche Methoden für die Klassifikation der Stellenanzeigen in Frage kommen. Nachdem in Kapitel 4 Möglichkeiten vorgestellt werden, wie eine Stellenanzeige für die Klassifikation verständlich repräsentiert werden kann, wird in Kapitel 5 die technische Umsetzung aller Komponenten in Java vorgestellt. Anschließend wird in Kapitel 6 analysiert wie gut die Umsetzung der Klassifikation funktioniert. Dazu muss zunächst erläutert werden, welche Evaluationsmaße es gibt und welche Aussage sie über die Effektivität der Klassifikation treffen können. Die Ergebnisse der Evaluation werden abschließend vorgestellt und diskutiert. Dabei werden mögliche Verbesserungsvorschläge angebracht.

2. Fragestellung

Die strukturierte Erfassung von Stellenausschreibungen steht seit einigen Jahren im Fokus verschiedener Forschungsbereiche. Da sich in den 90er Jahren internetbasierte Jobportale entwickelt haben, liegt eine große digitale Menge textueller Daten vor, die beispielsweise im Hinblick auf Qualifikationen von Arbeitnehmern oder auf Berufsfelder, in denen Arbeitskräftemangel herrscht, analysiert werden kann (Bensberg, F. und Buscher, G. (2016): 816). Durch die Aufbereitung der Stellenanzeigen können so Prognosen darüber gestellt werden, welche

Qualifikationen von Arbeitnehmern zukünftig verlangt werden, um Schulungen besser vorbereiten zu können. Qualifikationsentwicklungsforschung wird unter anderem im Bundesinstitut für Berufsbildung (BIBB) betrieben, weshalb in Kooperation mit der Universität zu Köln ein Framework entwickelt wurde, das Stellenausschreibungen in Absätze unterteilt und diese klassifiziert – der sogenannte JASC (*Job Ad Section Classifier*) (Hermes, J. und Schandock, M. (2016)). Da der in dieser Arbeit beschriebene Ansatz auf Teilen des JASC beruht, wird diese Implementation in Kapitel 4 näher erläutert. Die hier präsentierte Implementation dient allerdings nicht der Analyse des Arbeitsmarktes, sondern hilft dabei, Stellenausschreibungen mit Metadaten anzureichern, um dadurch relevante Stellen in einem internetbasierten Jobportal schneller ausfindig machen zu können. Ein Portal der get in GmbH, das sich auf Stellenausschreibungen für Jobeinsteiger in IT-Berufen spezialisiert[1], strukturiert bereits Anzeigen in diesem Bereich, um sie für Arbeitsuchende besser auffindbar zu machen. Allerdings werden die Kategorien, in die die Stellenanzeigen unterteilt werden können, bisher von Hand annotiert. Eine Implementation, welche die händische Arbeit durch maschinelle Lernverfahren unterstützen kann, ist daher erstrebenswert, um die bisherige Tätigkeit zu erleichtern.

2.1. Beschreibung der Daten

Zur Entwicklung einer Methode, die Stellenausschreibungen maschinell klassifiziert, stellte die get in GmbH eine Sammlung zur Verfügung, die aus 3362 Stellenanzeigen besteht, welche aus der Datenbank des Jobportals stammen und bereits klassifiziert sind. Diese Sammlung wird im Folgenden als Trainingsdaten bezeichnet und liegt im .xlsx-Format vor. Dabei repräsentiert eine Zeile jeweils eine Anzeige, zu der folgende Informationen vorliegen:

title	content	StudySubjects	ThematicPriorities	Degree

Die Einträge in *title* und *content* beinhalten die ursprüngliche Stellenausschreibung. Dabei liegt der Inhalt der Anzeige in html-Tags vor. Die übrigen drei Felder stellen Informationen bereit, die händisch aus der Stellenausschreibung extrahiert wurden. *StudySubjects* enthält alle Studienfächer, die in der Stellenanzeige aus Voraussetzung genannt werden, jeweils mit einem Komma getrennt. Dabei wird aus den folgenden Studienfächern ausgewählt: Wirtschaftsinformatik, Informatik, Geoinformatik, Bioinformatik, Mathematik, Wirtschaftsmathematik, Phy-

[1] https://www.get-in-it.de/ (zuletzt aufgerufen: 27.09.2017)

sik, BWL, Kommunikationswissenschaften, Wirtschaftsingenieurwesen, Marketing, Maschinenbau, Elektrotechnik, Informationstechnik und Softwaretechnik. *ThematicPriorities* umfasst – ebenfalls mit Kommata getrennt – alle von der get in GmbH zugeteilten Schwerpunkte. Da es sich ausschließlich um Stellenanzeigen aus dem IT-Bereich handelt, sind die Schwerpunkte inhaltlich auf diesen Bereich beschränkt. Die Labels wurden von der get in GmbH entwickelt und lauten wie folgt: Projektmanagement, Webentwicklung, Quality Assurance, Administration, Anwendungsentwicklung, Beratung / Consulting, Business Analysis, Datenbankentwicklung / BI, Risk / Compliance Management, Produktmanagement, System Engineering und Forschung & Entwicklung. In der Spalte *Degree* werden alle Studien- bzw. Ausbildungsabschlüsse aufgezählt, die in der Stellenanzeige als mögliche Job-Voraussetzung genannt werden.

Ohne zu diesem Zeitpunkt schon genauer auf die Implementation einzugehen, lassen sich mithilfe der Klasse `DataAnalyzeApp` erste Aussagen über die Struktur der Trainingsdaten machen. Tabelle 1 spiegelt die Häufigkeitsverteilungen von Studienfächern, Schwerpunkten und Abschlüssen pro Stellenausschreibung wider. Durchschnittlich ist eine Stellenanzeige also mit 2,96 Studienfächern, 1,66 Schwerpunkten und 2,31 möglichen Abschlüssen annotiert.

Anzahl Labels	**Studienfächer**	**Schwerpunkte**	**Abschlüsse**	**Anzahl Labels**	**Studienfächer**	**Schwerpunkte**	**Abschlüsse**
1	485	2052	199	**9**	6	-	-
2	1297	1039	2177	**10**	2	-	-
3	919	375	1232	**11**	1	-	-
4	489	148	46	**12**	1	-	-
5	211	34	-	**13**	1	-	-
6	140	4	-	**14**	40	-	-
7	44	2	-	**15**	1	-	-
8	17	-	-	**Arithmetisches Mittel**	2,96	1,66	2,31

Tabelle 1: Häufigkeitsverteilung von Labels pro Stellenausschreibung

In der vorliegenden Arbeit ist vor allem die Klassifikation der Schwerpunkte von Interesse. Deshalb werden im Folgenden die Häufigkeiten der einzelnen Schwerpunkte, die in Abbildung 1 zu sehen sind, näher beleuchtet. Die genauen Werte lassen sich der Tabelle 3 im Anhang entnehmen. Das vorliegende Diagramm zeigt die stark variierende Häufigkeit der Schwerpunkte. Stellenanzeigen, die mit dem Schwerpunkt „Anwendungsentwicklung“ markiert wurden, stechen vor allen anderen durch ihr häufiges Vorkommen heraus und umfassen 54 % der

gesamten Trainingsdaten. Gegensätzlich dazu sind die Schwerpunkte „Forschung & Entwicklung“, „Produktmanagement“ sowie „Risk / Compliance Management“ jeweils in nicht mehr als 3 % der Stellenanzeigen enthalten.

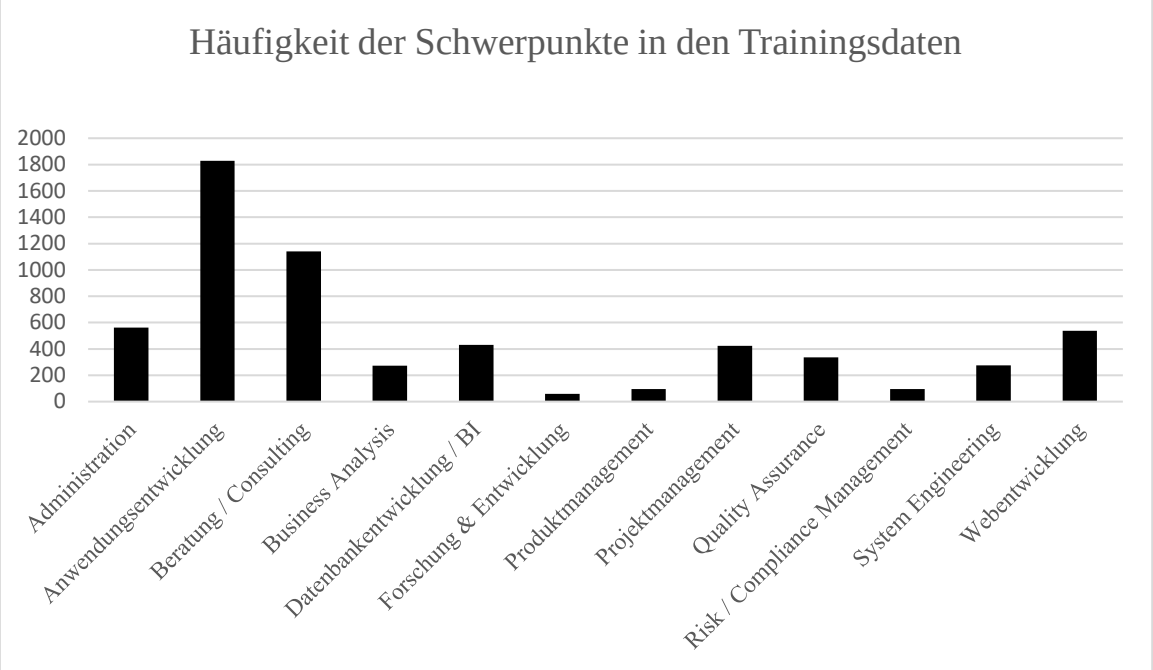

Abbildung 1 (Häufigkeit der Schwerpunkte)

Um für die Implementation festzulegen, welche Schwerpunkte klassifiziert werden sollen, bedarf es einer Datei im .xlsx-Format, die die Informationen dazu bereitstellt. Diese enthält in der ersten Spalte alle Schwerpunkte, die es gibt. Soll ein Schwerpunkt nicht in die Klassifikation einfließen, weil er beispielsweise nicht genügend Trainingsdaten besitzt, kann dieser mit einem vorangestellten Sternchen (*) gekennzeichnet werden. In der zweiten Spalte können Schlagwörter – jeweils mit einem Komma getrennt – angegeben werden, die im Inhalt einer Stellenanzeige mit entsprechendem Schwerpunkt vorkommen können.

2.2. Ziele

Die vorliegende Implementation wurde entwickelt, um die genannten Informationen, die bisher per Hand annotiert wurden, mittels maschinellem Lernens zu ermitteln. Der Fokus der Arbeit liegt dabei auf Lernverfahren, die für die Extraktion der Schwerpunkte angewendet werden können. Da sowohl die Abschlüsse als auch die Studienfächer über Wortabgleiche extrahiert werden können, wird dieser Teil der Implementation im Folgenden nicht weiter thematisiert. Die Implementation soll zum einen dazu dienen, für noch nicht gesichtete Stellenausschreibun-

gen Schwerpunkte vorauszusagen. Zum anderen können bereits händisch annotierte Stellenausschreibungen evaluiert werden, um gegebenenfalls Fehler in der händischen Klassifikation zu entdecken.

3. Klassifikation

Nicht nur im Bereich der Stellenausschreibungen werden Verfahren zur Klassifikation eingesetzt. Durch die immer größer werdende Sammlung von Dokumenten im Internet entwickelten sich in den 90er Jahren in der Computerlinguistik einige Methoden, um natürlichsprachliche Texte in vordefinierte Kategorien zu sortieren. Dies sollte dazu dienen relevante Informationen besser ausfindig zu machen. Beispielsweise konnten dadurch Zeitungsartikel in Rubriken wie Sport, Wirtschaft, Kultur etc. unterteilt werden (Carstensen, K.-U. et al. (2004): 496).

Grundsätzlich geht es bei Textklassifikation immer darum, boolesche Werte für jedes Paar $\langle d_j, l_j \in D \times L \rangle$ zu finden, wobei D alle vorhandenen Dokumente umfasst und L ein Set darstellt, dass alle zuvor festgelegten Labels, die jeweils für eine Kategorie stehen, beinhaltet. Dabei bedeutet *true*, dass Dokument d_j das Label l_j erhält und *false*, dass es nicht zugeschrieben wird. Die Labels haben einen rein symbolischen Charakter, da sie die Dokumente beschreiben und nicht als zusätzliche Information angesehen werden sollen. Des Weiteren darf für die Zuweisung der Labels zu den Dokumenten nur das endogene Wissen verwendet werden. In der reinen Klassifikation liegt also nur die Information, die der Text vermittelt, vor und diese Information wird nicht beispielsweise mit Ontologien angereichert. Allerdings können Klassifikationen in der Anwendung durch andere Methoden unterstützt werden, um so bessere Ergebnisse zu erzielen (Sebastiani, F. (2002): 3). Auf mögliche Kombinationen von Ansätze wird in Kapitel 7 eingegangen.

Die Klassifikation von Texten wurde zunächst in den 80er Jahren als Aufgabe der Wissensmodellierung angesehen. Dabei definierte man manuell eine Menge von Regeln, die Texte in die jeweiligen Kategorien zuordnen sollten. Da der Umfang der Texte allerdings während der 90er Jahre durch die Verbreitung des Internets stetig stieg, wurde es zunehmend schwieriger, Regeln zu erstellen, die auf den größer werdenden Korpus anwendbar waren. Es mussten daher Systeme entwickelt werden, die mit der kontinuierlich wachsenden Komplexität der Anforderungen umgehen konnten. Seitdem lässt sich Textklassifikation zusätzlich zum Bereich des Information Retrievals ebenso im *Machine Learning* verorten, da es einerseits darum geht, Informationen auffindbar zu machen und dies andererseits mithilfe von maschinellen Lernverfahren erreicht werden kann (Sebastiani, F. (2002): 2).

Textklassifikations-Systeme bestehen üblicherweise aus einem Model und der eigentlichen Klassifikation. Das Model dient dazu, Wissen aus bereits klassifizierten Texten zu erwerben, sodass für jede Kategorie, die zu klassifizieren ist, ein Profil erstellt werden kann. Dabei werden für alle Texte im Korpus Merkmale und Gewichtungen generiert, die den jeweiligen Text repräsentieren. Da die meisten Klassifikations-Systeme mit nummerischen Werten arbeiten, müssen also die natursprachlichen Daten in nummerische Merkmale umgewandelt werden (Carstensen, K.-U. et al. (2004): 496). Wie dies konkret geschieht, wird in Kapitel 4 erläutert. Carstensen et al. unterscheiden Textklassifikations-Systeme in zwei Dimensionen. Zunächst kann die Generierung der Kategorien-Profile auf lernende bzw. nicht-lernende Weise geschehen. In nicht-lernenden Systemen werden die Merkmale für jede Kategorie manuell zusammengestellt. Beispielsweise könnten bei der Klassifikation von Möbeln die Höhe oder das Material Merkmale sein, die Aufschluss darüber geben, ob es sich um einen Tisch, einen Stuhl oder ein Sofa handelt. Im Gegensatz dazu suchen lernende Systeme sich selbst relevante Merkmale aus und gewichten diese anschließend. Durch die Auswahl und die Gewichtung wird versucht, die Berechnungen, die zur Klassifikation führen, möglichst gering zu halten, da Merkmale, die keine Aussage über eine Klassenzugehörigkeit machen können, nicht in die Klassifikation einfließen (Carstensen, K.-U. et al. (2004): 497). Welche Methoden in der vorliegenden Implementation zur Merkmalsauswahl und -gewichtung verwendet werden, wird ebenfalls in Kapitel 4 näher erläutert. In der zweiten Dimension wird die Art der Klassifikation selbst definiert. Dort unterscheiden Carstensen et al. zwischen regelbasierten und statistischen Verfahren, die im Folgenden beschrieben werden.

3.1. Unterschiede zwischen regelbasierter und statistischer Klassifikation

In regelbasierten Verfahren werden zunächst Regeln festgelegt, anhand derer ein Text in Kategorien eingeordnet wird. Diese können manuell von Experten oder automatisch durch Entscheidungsbäume entworfen werden. Bei den Regeln handelt sich um boolesche Ausdrücke, die der Text erfüllen muss, um einer bestimmten Kategorie zu entsprechen. Beispielsweise kann geprüft werden, ob Wörter im vorliegenden Dokument enthalten sind, die das Auswahlkriterium für eine bestimmte Kategorie darstellen. Die Erzeugung eines solchen Models ist sehr zeitaufwändig, da Merkmale gefunden werden müssen, die die Kategorien klar voneinander unterscheiden. Außerdem droht ein sogenanntes *Overfitting*, sobald die Regeln zu spezifisch formuliert sind und nur noch auf einen bestimmten Text zutreffen können. Ein Vorteil von regelbasierten Verfahren ist allerdings, dass sie nicht zwingend Trainingsdaten benötigen, da die Regeln von Experten manuell erstellt werden können (Carstensen, K.-U. et al. (2004): 498).

Im Gegensatz dazu benötigen statistische Verfahren immer eine Menge an Trainingsdaten. Das hier entstehende Model wird aus den vorliegenden Daten generiert und ist deshalb wesentlich dynamischer als Modelle aus regelbasierten Verfahren. Ein Beispiel für ein statistisches Verfahren ist der *k-Nearest-Neighbour*-Algorithmus (kurz *kNN*). Dieser berechnet für jedes Dokument die *k* ähnlichsten Dokumente. Dies geschieht, indem jedes Dokument durch einen Merkmalsvektor repräsentiert wird, der aus den im Model berechneten Gewichten besteht. Die Distanz zwischen zwei Vektoren kann dabei über Maße wie die euklidische Distanz (Manning, C. D., Raghavan, P., und Schütze, H. (2009): 131) oder die Kosinus-Ähnlichkeit (Jurafsky, D. und Martin, J. H. (2009): 803) ermittelt werden. Anschließend können die Labels der nächsten Nachbarn Aufschluss über die Labels des betrachteten Dokuments geben. Beispielsweise geschieht dies über eine einfache Mehrheitsentscheidung (Carstensen, K.-U. et al. (2004): 499). Für die vorliegende Implementation wurde eine Abwandlung des *kNN* verwendet, auf welche in Kapitel 3.2.1. eingegangen wird.

Des Weiteren lassen sich im Bereich des maschinellen Lernens überwachte Verfahren von unüberwachten Verfahren unterscheiden. Da für die vorliegende Implementation bereits eine Menge an händisch annotierten Stellenanzeigen vorliegt, handelt es sich bei der Klassifikation um überwachtes Lernen, bei dem die klassifizierten Schwerpunkte mit dem Goldstandard, also den erwünschten Schwerpunkten, verglichen werden können. Unüberwachtes Lernen tritt im Gegensatz dazu bei Clustering-Verfahren auf, da die entstehenden Gruppen vorher nicht bekannt sind (Feldman, R. und Sanger, J. (2008): 70).

3.2. Unterschiede zwischen Multilabel- und Singlelabel-Klassifikation

Neben Unterschieden in der Art der Klassifikation selbst nimmt ebenso die Art der zu klassifizierenden Daten Einfluss auf das Verfahren. Die bisher beschriebenen Methoden bezogen sich auf den Fall, dass aus der Menge *L* genau ein Label für jedes Dokument ausgewählt wird, welches das Dokument am besten repräsentiert. Sobald die Menge *L* nur ein Label enthält, spricht man von einer binären Klassifikation. Ein bekanntes Beispiel dafür ist der Spam-Filter. Hier wird für jede E-Mail geprüft, ob sie in Bezug auf das Label „Spam“ eher als *true* oder eher als *false* eingestuft werden kann. Sobald $|L| > 1$ spricht man von einer *Multiclass*- bzw. Singlelabel-Klassifikation. Hier handelt es sich bei den klassifizierten Elementen um disjunkte Mengen, wobei jedem Element genau ein Label zugesprochen wird. Allerdings kann dies in den meisten Anwendungsfällen nicht umgesetzt werden, da sich Kategorien in vielen Bereichen nicht klar voneinander trennen lassen. Beispielsweise gibt es viele Filme, für die nicht nur ein

Genre zutrifft, und Musik kann mehrere Emotionen gleichzeitig auslösen (Spyromitros E., Tsoumakas G., und Vlahavas I. (2008)). Auch Zeitungsartikel betreffen häufig mehr als ein Thema. In diesen Fällen wir einem Dokument d_j nicht nur ein Label l_j aus den zur Verfügung stehenden Labels L hinzugefügt. Vielmehr wird ein Set aus L ausgewählt, weshalb man auch von einer sogenannten Multilabel-Klassifikation spricht (Herrera, F. et al. (2016): 11-13). Dies trifft auch auf die vorliegende Klassifikation von Stellenausschreibungen zu. Zur Beschreibung von Datenmengen, deren Elemente durch mehr als ein Label repräsentiert werden, dienen die Maße der Label-Kardinalität und der Label-Dichte. Die Label-Kardinalität bezeichnet die durchschnittliche Anzahl der Labels pro Element der Datenmenge. Im Fall der Stellenanzeigen beträgt diese 1,66 Schwerpunkte pro Anzeige (siehe Tabelle 1). Die Label-Dichte relativiert diesen Wert an der Anzahl der möglichen Labels. In unserem Beispiel herrscht also eine Dichte von 0,14 (Tsoumakas, G. und Katakis, I. (2008): 70).

Solange nur wenige Kombinationen von Labels in den zu klassifizierenden Elementen vorkommen, ist es möglich, diese Kombinationen wiederum in eigene Labels umzuwandeln. Diese Methode gehört zur Gruppe der sogenannten Problemtransformationen und wurde im Rahmen des JASC angewandt. Zusätzlich zu den reinen vier Labels wurden zwei weitere Labels definiert, die jeweils eine Kombination aus zwei der ursprünglichen Labels darstellen (Hermes, J. und Schandock, M. (2016): 12). Im Fall der vorliegenden Daten ist dies allerdings nicht zielführend, da dort 249 verschiedene Kombinationen auftreten, sodass nicht für jede Kombination genügend Trainingsdaten vorhanden wären. Außerdem könnten bei neuen Stellenanzeigen Kombinationen auftreten, die noch nicht in den bereits bekannten Kombinationen auftreten. Algorithmen, die als Problemtransformationen bezeichnet werden, gehen im Allgemeinen so vor, dass sie die vorliegenden Daten so weit verändern, dass die Anwendung eines Singlelabel-Klassifikators möglich wird. Die soeben beschriebene Methode, für jede Kombination ein neues Label zu kreieren, verwandelt die Multilabel-Klassifikation in eine *Multiclass*-Klassifikation. Eine andere Möglichkeit, das Problem zu transformieren, besteht darin, alle Labels gesondert voneinander zu betrachten und so für jedes Label eine binäre Klassifikation vorzunehmen. Dabei wird für jedes Label ein eigenes Model erstellt. Der Klassifikator gibt, basierend auf dem jeweiligen Model, ein *true* aus, sobald das dazugehörige Label für das Set des Dokuments vorausgesagt wird, und *false*, wenn dies nicht der Fall ist (Herrera, F. et al. (2016): 66). Die Anwendung einer Problemtransformation wird in Kapitel 3.2.2. beschrieben.

Neben der Problemtransformation gibt es die Möglichkeit der Algorithmus-Adaption. Statt die Daten an einen Algorithmus anzupassen werden dabei bekannte Algorithmen aus dem Bereich

der Singlelabel-Klassifikation so umformuliert, dass sie pro Label eine Voraussage machen können (Herrera, F. et al. (2016): 81). Beispielsweise können mithilfe des *ML-C4.5* Entscheidungsbäume entwickelt werden, die mit mehreren Labels pro Element arbeiten können. Ebenso ist es möglich, *Support Vector Machines* (SVM) an die Problematik der Multilabel-Klassifikation anzupassen. Da SVMs im herkömmlichen Sinne für die binäre Klassifikation geeignet sind und daher nicht mit der Abhängigkeit zwischen Labels umgehen können, gibt es Anwendungen wie beispielsweise Rank-SVM, die als Multilabel-Klassifikator eingesetzt werden können (Elisseeff, A., Weston, J. (2002)). Im Gegensatz zu den genannten Algorithmus-adaptierenden Klassifikatoren benötigen sogenannte Instanz-basierte Methoden kein Model, um klassifizieren zu können. Vielmehr geht es darum, neue Instanzen aufgrund von Trainingsinstanzen zu klassifizieren, die der Testinstanz ähnlich sind (Mooney, R. J. (2005): 384f.). Ein Beispiel dafür ist *Multilabel kNN* (kurz: *ML-kNN*), eine Abwandlung des bereits in 3.1. erwähnten *kNN*-Algorithmus. Hier werden Annahmen für die Kategorien der neuen Instanz aufgrund der Kategorien ihrer *k* nächsten Nachbarn angestellt.

3.2.1. *ML-kNN*

Die vorliegende Implementation enthält unter anderem eine Umsetzung des von Zhang und Zhou entwickelten Algorithmus *ML-kNN* (Zhang, M.-L. und Zhou, Z.-H. (2007)), der dazu dient, dem Element t ein Set aus der zur Verfügung stehenden Label-Menge L zuzuordnen. Um zu entscheiden, welche Labels für dieses Set ausgewählt werden, zieht *ML-kNN* statistische Informationen aus den Trainingsdaten heran. Genauer genommen sind diese Informationen bedingte Wahrscheinlichkeiten, die mithilfe der Trainingsdatenmenge approximiert werden. Dafür benötigt der Algorithmus die folgenden Parameter: einen Merkmalsvektor, der das zu klassifizierende Element t repräsentiert, die Menge der Trainingsdaten, einen Glättungsparameter s, sowie die Ganzzahl k, die festlegt, wie viele Nachbarn in die Berechnung mit einbezogen werden sollen. Die Ausgabe besteht aus einem Vektor $\overrightarrow{y_t}$ der Länge $|L|$, der für jedes Label, das prognostiziert wird, entweder eine 1 oder eine 0 enthält, je nachdem, ob das jeweilige Label für t ausgewählt wurde oder nicht (Zhang, M.-L. und Zhou, Z.-H. (2007): 2040). Des Weiteren liefert der Algorithmus als Ausgabe einen Vektor $\overrightarrow{r_t}$. Die Länge dieses Vektors beträgt ebenfalls die Anzahl der Labels, allerdings enthält er für jedes Label die bedingte Wahrscheinlichkeit, die für dieses Label berechnet wurde. So lässt sich ablesen, welches Label das Element am besten repräsentiert.

Wie beim herkömmlichen *kNN*-Algorithmus werden auch hier die *k* nächsten Elemente mithilfe der Distanz zwischen den Merkmalsvektoren der Elemente ermittelt. Allerdings wird ein Label

nun nicht einfach vergeben, wenn die Mehrheit der Nachbarn es trägt. Vielmehr zieht der Algorithmus wie bereits erwähnt statistische Informationen heran, die er zuvor aus den gesamten Trainingsdaten gewonnen hat. Dies geschieht mithilfe eines Zählvektors $\overrightarrow{C_x}$, der wie folgt definiert ist:

$$\overrightarrow{C_x}(l) = \sum_{a \epsilon N(x)} \overrightarrow{y_a}(l), \qquad l \in L$$

$N(x)$ enthält dabei alle k nächsten Nachbarn des Elements x, wobei x ein Element der Trainingsdaten darstellt. Es wird also für jedes Label l in L gezählt wie viele der k Nachbarn von x dieses Label tragen. Gleichzeitig wird festgestellt, ob x selbst durch l repräsentiert wird. Diese Information wird auf den Arrays c und c' , die jeweils k Elemente enthalten, gespeichert. Dabei gilt folgendes Schema: sollte x das Label besitzen und j seiner Nachbarn ebenfalls, so wird der Wert $c[j]$ um 1 erhöht. Sollte das Label l das Element x nicht beschreiben, so wird stattdessen $c'[j]$ um 1 erhöht (Zhang, M.-L. und Zhou, Z.-H. (2007): 2041). So enthalten beide Arrays Informationen darüber, inwiefern die Anzahl der Nachbarn mit einem bestimmten Label ein Indiz dafür ist, dass das betrachtete Element selbst ebenfalls das Label besitzt oder dass er es nicht besitzt.

Nachdem mithilfe aller Elemente in den Trainingsdaten diese statistischen Werte erhoben wurden, soll für eine Testinstanz t vorausgesagt werden, welche Labels sie repräsentieren. Dabei stellt H_1^l das Ereignis dar, dass das betrachtete Element t selbst das Label l besitzt, H_0^l steht hingegen für das Komplementärereignis. Auch für t werden zunächst die Labels der benachbarten Elemente gezählt. Dabei bezeichnet $E_j^l (j \in \{0,1, \dots, k\})$ das Ereignis, dass exakt j Nachbarn das Label l besitzen. Anschließend werden E_j^l und H_1^l bzw. E_j^l und H_0^l miteinander in Beziehung gesetzt. Ziel dabei ist es, herauszufinden, ob H_1^l oder H_0^l unter dem Ereignis E_j^l wahrscheinlicher ist. Da das Ziel also ist, bedingte Wahrscheinlichkeiten zu ermitteln, verwenden Zhang und Zhou für diese Berechnung den Satz von Bayes. Für Ereignisse A und B sagt dieser aus:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Die Wahrscheinlichkeit von Ereignis A unter der Bedingung von Ereignis B kann also berechnet werden, wenn die A-priori-Wahrscheinlichkeiten der Ereignisse A und B sowie die Wahrscheinlichkeit von Ereignis B unter der Bedingung von Ereignis A bekannt sind (Carstensen,

K.-U. et al. (2004): 118). Im Fall von *ML-kNN* werden jeweils die Wahrscheinlichkeit von H_1^l unter der Bedingung von E_j^l sowie die Wahrscheinlichkeit von H_0^l unter der Bedingung von E_j^l berechnet. Danach entscheidet das Maximum der beiden Wahrscheinlichkeiten, ob das Label gesetzt wird oder nicht. In Worte gefasst lautet also an dieser Stelle die Frage: Ist die Wahrscheinlichkeit, dass Element t das Label l erhält, größer als die Wahrscheinlichkeit, dass dies nicht eintritt, wenn j von k Nachbarn ebenfalls das Label l besitzen? Dies bedeutet in Bezug auf den Satz von Bayes, dass Ereignis A das Setzen bzw. Nicht-Setzen eines Labels bezeichnet. Ereignis B hingegen steht für das Eintreten einer exakten Anzahl von Nachbarn mit diesem Label. Da Ereignis B unabhängig davon ist, ob das Label gesetzt wird oder ob es nicht gesetzt wird, kann der Nenner der Gleichung in diesem Fall weggelassen werden. Mathematisch ausgedrückt ergibt dies:

$$\overrightarrow{y_t}(l) = \underset{b\in\{0,1\}}{\operatorname{argmax}} P\left(H_b^l \middle| E_j^l\right) = \underset{b\in\{0,1\}}{\operatorname{argmax}} \frac{P\left(E_j^l \middle| H_b^l\right) * P(H_b^l)}{P(E_j^l)} = \underset{b\in\{0,1\}}{\operatorname{argmax}} P\left(E_j^l \middle| H_b^l\right) * P(H_b^l)$$

$\overrightarrow{y_t}$ erhält demnach für das betrachtete Label den Wert, bei dem diese Gleichung maximiert wird. $P(H_b^l)$ stellt die A-priori-Wahrscheinlichkeit für das Auftreten bzw. das Nicht-Auftreten des Labels dar. Diese berechnen Zhang und Zhou folgendermaßen[2]:

$$P\left(H_1^l\right) = \frac{s + \sum_{i=1}^{m} \overrightarrow{y_{x_i}}(l)}{s * 2 + m} \qquad P\left(H_0^l\right) = 1 - P\left(H_1^l\right)$$

wobei m die Anzahl der Elemente im Trainingsdatensatz darstellt.

$P\left(E_j^l \middle| H_b^l\right)$ lässt sich mithilfe der zuvor gebildeten Arrays c und c' errechnen. Zhang und Zhou approximieren aus den Trainingsdaten die beiden bedingten Wahrscheinlichkeiten für j Nachbarn mit dem betrachteten Label wie folgt:

$$P\left(E_j^l \middle| H_1^l\right) = \frac{s + c[j]}{s * (k+1) + \sum_{p=0}^{k} c[p]}$$

$$P\left(E_j^l \middle| H_0^l\right) = \frac{s + c'[j]}{s * (k+1) + \sum_{p=0}^{k} c'[p]}$$

[2] vgl. Zhang, M.-L. und Zhou, Z.-H. 2007: 2041, Pseudocode Zeile 2

wobei s zunächst auf 1 gesetzt wird. Der Wert für $\overrightarrow{r_t}(l)$ entsteht schließlich aus der Relation zwischen den beiden berechneten Wahrscheinlichkeiten:

$$\overrightarrow{r_t}(l) = \frac{P(H_1^l) * P\left(E^l_{\overrightarrow{C_t}(l)} | H_1^l\right)}{\sum_{b\in\{0,1)} P(H_b^l) * P\left(E^l_{\overrightarrow{C_t}(l)} | H_b^l\right)}$$

Mithilfe des Ranking-Vektors lassen sich die Labels untereinander im Hinblick auf ihre Relevanz für das Element t vergleichen.

3.2.2. *Naive Bayes*

Um die Effektivität von *ML-kNN* richtig einschätzen zu können, wurde in der Implementation außerdem ein *Naive Bayes*-Klassifikator eingesetzt. Die Klassifikation erfolgt hier auf der Grundlage einer Problemtransformation, es wird also jedes Label separat klassifiziert. *Naive Bayes*-Klassifikatoren basieren auf dem bereits vorgestellten Satz von Bayes und können dafür verwendet werden, zwei Hypothesen gegeneinander abzuwägen. Der Klassifikator wird deshalb als naiv bezeichnet, weil er davon ausgeht, dass alle Merkmale des zu klassifizierenden Elements unabhängig voneinander sind und direkten Einfluss auf die Zugehörigkeit zu einer Klasse haben (Russell, S. J. (1995): 808). Im Gegensatz zum *ML-kNN* tragen nun nicht die Eigenschaften der Nachbarn, sondern die Merkmale des Elements selbst zur Klassifikation bei. Um also zu entscheiden, ob ein Element x das Label l tragen soll, wird folgende Wahrscheinlichkeit berechnet:

$$P(l|x_1, \dots\dots, x_n) = P(l) * \prod_i P(x_i|l)$$

$P(l)$ stellt wie im Fall von *ML-kNN* die a-priori-Wahrscheinlichkeit für das Label l dar. Die Wahrscheinlichkeit $P(x_i|l)$ kann mithilfe der Trainingsdaten approximiert werden, indem gezählt wird, wie oft ein Merkmal x_i in Elementen mit dem Label l vorkommt. Anschließend wird die Wahrscheinlichkeit $P(\neg l|x_1, \dots\dots, x_n)$ errechnet, indem ebenfalls in den Trainingsdaten gezählt wird, wie oft die jeweiligen Merkmale in Elementen ohne das Label l vorkommen. Das Maximum der beiden bedingten Wahrscheinlichkeiten entscheidet schließlich, ob das Label gesetzt wird. Außerdem kann äquivalent zu *ML-kNN* die tatsächliche Wahrscheinlichkeit für den Rankingvektor aus der Relation zwischen den beiden Wahrscheinlichkeiten errechnet werden. Dieser Vorgang wird für jedes Label in L wiederholt.

3.3. Anwendungsgebiete

Neben *ML-kNN* und Naive Bayes gibt es viele Lösungen, mit deren Hilfe man einem Objekt mehrere Labels zuschreiben kann. Ebenso divers fallen die Bereiche aus, in denen Elementen mehrere Kategorien zugeschrieben werden sollen. Zhang und Zhou testen ihren Algorithmus beispielsweise an funktionellen Genomanalysen von Hefe, an natürlichen Kulissen in Bildern sowie an einer Kategorisierung von Internetseiten (Zhang, M.-L. und Zhou, Z.-H. (2007): 2041-47). Dadurch zeigt sich, dass *ML-kNN* nicht an eine Domäne gebunden ist. Zu den großen Bereichen, die Multilabel-Klassifikation gebrauchen, zählt vor allem die Kategorisierung von Texten. Dieses Feld kann als Startpunkt der Multilabel-Klassifikation angesehen werden, nicht nur wegen der Masse an textuellen Daten im Internet, die durch Klassifikation strukturiert werden können. Auch außerhalb des World Wide Web liegen Mengen von unstrukturierten Daten, wie beispielsweise elektronischen Dokumenten, vor, die durch Klassifikation erfasst werden sollen (Herrera, F. et al. (2016): 20). Unter anderem nennt Sebastiani als Anwendungen im Bereich der Textklassifikation die Filterung von Dokumenten, die Generierung von Metadaten sowie die Disambiguierung von Wörtern (Sebastiani, F. (2002): 1).

Des Weiteren bietet die Strukturierung multimedialer Ressourcen viele Optionen, Multilabel-Klassifikation einzusetzen. Dieser Bereich verlangt vor allem aufgrund der Entwicklung des Internets und der immer größer werdenden Menge an Videos, Musik und Bildern nach Anwendungen, diese Medien zu kategorisieren und dadurch leichter auffindbar zu machen. Ein Beispiel in diesem Bereich ist die automatisierte Ermittlung von Ereignissen in Videos (Chen, X.-j. et al. (2016)). Da hier meist mehrere Ereignisse gleichzeitig stattfinden, soll ein Algorithmus dafür sorgen, Elemente im Video zu identifizieren und einem semantischen Konzept zuzuordnen.

Als weiteres Anwendungsgebiet wurde in Bezug auf Zhang und Zhou bereits ein Beispiel aus dem Bereich der Biologie genannt. Dieser stellt, insbesondere durch die Genetik, eine Disziplin dar, die ebenfalls viele Anwendungsfälle für Multilabel-Klassifikation anbietet. Im Bereich der Genetik sind zudem häufig hierarchische Multilabel-Klassifizierungsmodelle notwendig. Dabei kann ein Element – zum Beispiel ein Gen – zu mehreren Klassen gehören, während diese Klassen wiederum zu anderen Klassen gehören und somit hierarchisiert sind (Vens, C. et al. (2008)). Weitere Anwendungen wie beispielsweise die Untersuchung nonverbaler Ausdrücke haben Herrera, Charte et all. zusammengefasst (Herrera, F. et al. (2016): 22).

3.3.1. Bezug zur Problemstellung

Die in dieser Arbeit vorgestellte Implementation soll Stellenanzeigen Schwerpunkte zu ordnen. Es handelt sich um eine Multilabel-Klassifikation, bei der die 12 möglichen Labels nicht hierarchisch angeordnet sind. Da das Problem aus dem Bereich der Textklassifikation stammt, werden für die Vorverarbeitung der Dokumente Methoden aus dem Information Retrieval verwendet. Die Selektion der Merkmale wird durch lernende Mechanismen realisiert, die im Folgenden vorgestellt werden. Für die eigentliche Klassifikation stehen der *ML-kNN*-Algorithmus sowie *Naive Bayes* zur Verfügung. Da diese beiden statistischen Ansätze wie viele andere Klassifikatoren domänenunabhängig sind, lassen sie sich auch im Bereich der Textklassifikation einsetzen.

4. Feature Engineering

Um eine Stellenanzeige in einen Vektor umzuwandeln, der von einem Klassifikator interpretiert werden kann, bedarf es eines sogenannten *Feature Engineerings*. Dabei geht es darum, Merkmale auszusuchen und diese nach einem bestimmten Prinzip zu gewichten, sodass der dadurch entstehende Merkmalsvektor die Stellenanzeige repräsentieren kann. Die Methoden, welche in der vorliegenden Implementation zur Auswahl stehen, werden hier zunächst erklärt. Wie bereits angeführt, lässt sich Klassifikation als eine Aufgabe des Information Retrieval betrachten, weshalb die Methoden des *Feature Engineerings* aus diesem Bereich stammen. Auch dort werden Dokumente im sogenannten Vektorraummodell dargestellt und müssen deshalb durch ausgesuchte Merkmale und deren Werte repräsentiert werden (Jurafsky, D. und Martin, J. H. (2009): 802).

Eine Stellenanzeige besteht, wie jeder andere natursprachliche Text, auf unterschiedlichen Ebenen aus aneinandergereihten Einheiten. Diese Einheiten reichen von Zeichen über Wörter und Phrasen bis hin zu ganzen Sätzen und Abschnitten (Mikheev, A. (2005): 201). Sie können als Features genutzt werden, um das jeweilige Dokument zu repräsentieren. Eine Möglichkeit ist es, die Dokumente im Korpus durch sogenannte *Bag-of-Words* zu repräsentieren. Dabei beträgt die Länge des Merkmalsvektors die Anzahl aller Wörter, die im gesamten Korpus vorkommen. Für ein bestimmtes Dokument nimmt der Vektor an der Position eines bestimmten Worts nun eine 1 an, wenn das Wort im Dokument vorkommt. Andernfalls wird eine 0 an der entsprechenden Position im Vektor gesetzt. Diese Methode ist leicht umzusetzen, allerdings umfassen selbst kurze Dokumente wie beispielsweise Nachrichtentexte viele verschiedene Wörter, sodass die Vektoren oftmals sehr lang werden (Feldman, R. und Sanger, J. (2008): 68). Um die Anzahl

der Features zu verringern gibt es Methoden, die dabei helfen relevante Einheiten aus dem Dokument zu selektieren.

4.1. Auswahl der Features

Um Features auszuwählen, die ein Dokument repräsentieren, muss zunächst festgelegt werden, was überhaupt als Feature gilt. Wie bereits vorgeschlagen, kann eine *Bag-of-Words* erstellt werden, die Wörter, die im Dokument enthalten sind, sammelt. Doch lässt sich an dieser Stelle darüber streiten, was nun genau als Wort erfasst werden soll. Im Information Retrieval gibt es eine Unterscheidung zwischen Tokens, Typen und Termen. Ein Token ist eine Instanz, die aus einer Folge von Symbolen besteht, und in einem Dokument zusammen mit anderen Token eine semantische Einheit darstellt. Ein Typ hingegen ist die Klasse, der ein Token angehört. Dabei besitzen alle Tokens einer bestimmten Klasse dieselbe Abfolge von Symbolen. Terme wiederum repräsentieren einen Typ in einem Wörterbuch (Manning, C. D., Raghavan, P., und Schütze, H. (2009): 22f.). Wie bereits angeführt bieten sich Tokens als Features nur bedingt an, da der Vektor in diesem Fall sehr lang wird und die Dokumente dadurch weniger vergleichbar werden. Beispielsweise werden hier ein „Beraten“ am Satzanfang und ein „beraten“ im Satzverlauf nicht als gleich identifiziert. Deshalb ist es sinnvoll, Methoden anzuwenden, die die Tokens in normierte Terme umwandeln, damit die Dokumente untereinander vergleichbar werden. Um das soeben angesprochene Beispielproblem zu lösen, können alle Tokens zunächst normalisiert werden, indem alle Großbuchstaben in Kleinbuchstaben umgewandelt werden. Eine weitere Möglichkeit, um den Vektor zu reduzieren, ist die Eliminierung von Füllwörtern. Damit sind Wörter gemeint, die in allen Texten häufig vorkommen und somit nicht zur inhaltlichen Aussage des Dokuments beitragen. Für die Filterung solcher Füllwörter kann eine einfache Liste verwendet werden, die Wörter enthält, welche nicht in die Erstellung des Merkmalsvektors einbezogen werden sollen. Des Weiteren können Tokens mithilfe eines sogenannten Stemmers auf ihren Wortstamm zurückgeführt werden. Dies macht es beispielsweise möglich verschiedene Deklinationen auf das gleiche Wort zurückzuführen. Die letzte in der Implementation vorhandene Möglichkeit, um die Anzahl der Merkmale zu reduzieren, ist die Zerteilung der Terme in sogenannte Buchstaben-N-Gramme. Dabei wird ein Term in Buchstabensequenzen der Länge *n* zerlegt, die schließlich die Merkmale im Vektor repräsentieren.

4.2. Gewichtung der Features

Nachdem Merkmale ausgewählt wurden, welche die Stellenanzeigen repräsentieren sollen, muss der Merkmalsvektor für ein Dokument mit Werten gefüllt werden. Eine einfache Möglichkeit ist es, die Anwesenheit oder Abwesenheit eines Merkmals durch eine 1 oder eine 0

abzubilden. Jedoch enthält der Merkmalsvektor dabei keine Information darüber, wie wichtig das Merkmal für die Stellenanzeige ist. Deshalb gibt es verschiedene Möglichkeiten, die Merkmale zu gewichten. In der Implementation können dafür das Tf-idf-Maß sowie die Log-Likelihood-Funktion verwendet werden. Bei Ersterem gilt zunächst die Annahme, dass ein Term, der in einem Dokument *a* häufiger vorkommt als in einem Dokument *b* für *a* relevanter ist. Deshalb kann der Merkmalsvektor für jedes Dokument zunächst die Termfrequenzen(tf) der einzelnen Merkmale bestehen. Da die Länge des Dokuments einen Einfluss auf die Frequenz eines Terms hat, gibt es die Möglichkeit, alle Frequenzen des Dokuments an der maximalen Frequenz des Dokuments zu normieren. Des Weiteren gelten Terme, die in einem Dokument häufig vorkommen, während sie in anderen Dokumenten gar nicht oder selten auftauchen, als besonders relevant für dieses Dokument. Deshalb lässt sich die Termfrequenz an der inversen Dokumentenfrequenz relativieren (Jurafsky, D. und Martin, J. H. (2009): 805). Dadurch ergibt sich für den Term w_i im Dokument j:

$$w_{i,j} = \frac{tf_{i,j}}{\max\limits_{i' \in j} tf_{i',j}} * \log \frac{N}{n_i}$$

Dabei stellt N die Summe aller Dokumente und n_i die Summe aller Dokumente, die w_i enthalten, dar. Eine weitere Methode, um die Relevanz eines Terms für ein Dokument zu berechnen, ist der Gebrauch einer Log-Likelihood-Gewichtung. Dabei wird das Gewicht für den Term w_i im Dokument j mithilfe eines Quotienten gebildet, der aus der Wahrscheinlichkeit von w_i in j und der Wahrscheinlichkeit von w_i im gesamten Korpus gebildet wird (vgl. Dunning, T. (1993)); (vgl. Zampieri, M., Hermes, J., und Schwiebert, S. (2013)).

5. Die Implementation

Im Folgenden wird nun erklärt wie die beschriebenen Methoden in der Anwendung umgesetzt wurden. Die vorliegende Anwendung, die Stellenanzeigen in Schwerpunkte kategorisiert, basiert auf der Struktur des bereits erwähnten JASC. Da die Aufgabe des JASC darin besteht, Stellenanzeigen aus einer Datenbank in Abschnitte zu unterteilen und diese zu klassifizieren, bot es sich an, dieses Framework zu erweitern, da viele Arbeitsschritte identisch sind.

Für den Anwender bietet die Implementation verschiedene Möglichkeiten, neue Stellenanzeigen zu klassifizieren oder bereits klassifizierte Stellenanzeigen zu analysieren und zu evaluieren. In den folgenden Abschnitten geht es zunächst um die Klassifikation neuer Stellenanzeigen, welche durch die Klasse `JobAdClassificationApp` realisiert werden kann. In einer

Konfigurationsdatei[3] müssen zunächst die Parameter für die Klassifikation festgelegt werden. In dieser Datei ist das Attribut jeweils durch einen Doppelpunkt vom Wert getrennt. Auf welche Variablen der Anwender Einfluss hat und welche Werte zulässig sind, kann aus Tabelle 5 im Anhang entnommen werden.

5.1. Klassifikation von Stellenausschreibungen

5.1.1. Input

Um die Trainingsdaten für die Klassifikation verfügbar zu machen, muss zunächst der Inhalt der Stellenanzeige aus der .xlsx-Datei eingelesen werden. Dabei wird für jede Zeile der Datei ein Objekt der Klasse `FocusClassifyUnit` erzeugt. Das Auslesen der Datenbank geschieht mithilfe der Methode `List<FocusClassifyUnit> getTrainingData(Boolean safeUnused)` der Klasse `TrainingUnitGenerator`, die sich der Apache-API *POI-XSSF*[4] bedient, um Daten aus einer Excel Tabelle auszulesen. Der Titel der Anzeige sowie ihr Inhalt werden zum gesamten Inhalt der Anzeige zusammengefasst, wobei zunächst die html-Tags im Text mithilfe von *jsoup*[5] entfernt werden. Anschließend wird der gesäuberte Text als Inhalt der Stellenanzeige im `FocusClassifyUnit`-Objekt gespeichert. Die Labels werden in einer `Map<String, Boolean>` gespeichert, wobei der Key *true* annimmt, sobald das Label für die Stellenanzeige vorhanden ist. Anschließend wird den Konfigurationsdaten entsprechend ein Merkmalsvektor für jede Stellenanzeige erzeugt.

5.1.2. Klassifikation

Nachdem die Trainingsdaten für die Klassifikation verfügbar sind, wird mit ihnen und dem entsprechenden Klassifikator ein Model erstellt. Im Fall von *ML-kNN* bedeutet dies, dass die bedingten Wahrscheinlichkeiten berechnet werden, die für die Klassifikation notwendig sind (siehe Kapitel 3.2.1.). Da es sich bei der Umsetzung von Naive Bayes um eine Problemtransformation handelt, besteht das Klassifikationsmodel aus mehreren kleinen Modellen, jeweils eins pro Schwerpunkt. Für jedes Label wird berechnet wie hoch die A-priori-Wahrscheinlichkeit des Labels ist und welche Merkmale in wie vielen der Trainingsdaten, die das Label tragen, enthalten sind.

Nachdem anschließend ebenfalls die Testdaten eingelesen und in Merkmalvektoren umgewandelt wurden, kann die eigentliche Klassifikation beginnen. Ist *ML-kNN* als Klassifikator gesetzt, so werden die Nachbarn der Testinstanz identifiziert und deren Labels gezählt. Aufgrund dieser

[3] ml_classification/configurations.txt
[4] https://poi.apache.org/spreadsheet/ (zuletzt aufgerufen: 27.09.2017)
[5] https://jsoup.org/ (zuletzt aufgerufen: 27.09.2017)

Häufigkeiten und der zuvor berechneten bedingten Wahrscheinlichkeiten wird für jedes Label eine eigene Wahrscheinlichkeit berechnet. Im Fall von Naive Bayes wird jedes Label separat klassifiziert. Dazu werden alle Merkmale der zu klassifizierenden Stellenanzeige herangezogen, sodass diese mit den Dokumentfrequenzen im vorliegenden Label verglichen werden können. Schließlich wird die daraus entstehende Wahrscheinlichkeit sowohl im Fall von *Naive Bayes* als auch bei *ML-kNN* mit dem in den Konfigurationen festgelegten Schwellwert verglichen. Daraufhin wird das Label entsprechend auf *true* oder *false* gesetzt. Die tatsächlich berechnete Wahrscheinlichkeit wird zusammen mit dem Label in die Ranking-Map der `FocusClassifyUnit` geschrieben.

Bei einer Multilabel-Klassifikation kann es passieren, dass bei einem Element kein Label gesetzt wird, weil alle Wahrscheinlichkeiten unter dem gesetzten Schwellwert liegen. In diesem Fall kann der Anwender über die Konfigurationsdatei eingreifen und festlegen, dass das wahrscheinlichste Label für die Stellenanzeige gesetzt werden soll. Dies ist allerdings nur für die Evaluation relevant sowie wenn die Ausgabe die gesetzten Labels und nicht das Ranking enthalten soll.

5.1.3. Output

Die Ausgabe der klassifizierten Stellenanzeigen erfolgt ebenfalls im .xlsx-Format im vom Anwender angegebenen Ordner. Die Tabelle hat die selbe Struktur wie die Datenbank der Trainingsdaten und die Eingangsdatei. Der Inhalt der Stellenanzeige wird inklusive html-Tags ausgegeben. Wie oben genannt kann der Anwender entschieden, ob die Schwerpunkte der klassifizierten Stellenausschreibung als gesetzte Labels oder als Ranking ausgegeben werden. Die Ausgabe der konkreten Wahrscheinlichkeiten hat den Vorteil, dass die Abstufung zwischen den einzelnen Schwerpunkten für den Anwender sichtbar wird. Allerdings hat das Feld *ThematicPriorities* so nicht die gleiche Struktur wie in den Trainingsdaten und müsste noch einmal händisch umgeformt werden, um beispielsweise im Jobportal genutzt werden zu können.

5.2. Datenanalyse

Wie bereits im Kapitel zur Beschreibung der Daten erwähnt, bietet die vorliegende Implementation zudem die Möglichkeit, die Trainingsdaten zu analysieren. Dies spielt vor allem für die Auswahl von weiteren Klassifikationsalgorithmen eine Rolle, da bei der Sichtung von bestehenden Algorithmen Werte wie die Label-Kardinalität oder die Label-Dichte eine Rolle spielen. Die ausführbare Klasse `DataAnalyseApp` erstellt aus der Trainings-Datenbank `FocusClassifyUnit`-Objekte, welche dann im Hinblick auf Label-Verteilungen bei

Schwerpunkten, Studienfächern und Abschlüssen sowie auf die Häufigkeit von Label-Kombinationen analysiert werden. Das Ergebnis der Analyse wird schließlich als .txt-Datei ausgegeben. Ein Teil der Ergebnisse wurde bereits in Kapitel 2.1. dargestellt.

6. Evaluation

Die Implementation bietet, wie bereits erwähnt, die Möglichkeit, die Klassifikation zu evaluieren. Dabei kann man entweder verschiedene Konfigurationen testen[6] und die Ergebnisse im zweiten Schritt ranken lassen oder die Klassifikation unter einer bestimmten Konfiguration evaluieren[7]. In beiden Fällen wird die Methode der sogenannten Kreuzvalidierung genutzt, die im folgenden Kapitel beschrieben wird. Anschließend werden die genutzten Evaluationsmaße vorgestellt sowie die Ergebnisse der Evaluation präsentiert.

6.1. Kreuzvalidierung

Die Kreuzvalidierung ist eine Methode, mit der man anhand von Trainingsdaten prüfen kann, wie gut ein Klassifikationsalgorithmus funktioniert. Dabei wird die Datenmenge zufällig in *k* gleichgroße Mengen zerteilt. Die Klassifikation wird nun an jeder dieser *k* Mengen getestet, jeweils mit den übrigen Mengen als Trainingsdaten. Dadurch kann sichergestellt werden, dass jedes Element der Trainingsdaten einmal klassifiziert wurde. Anschließend werden die klassifizierten Labels mit den in den Trainingsdaten vorhandenen Labels für das betrachtete Element verglichen, wobei verschiedene Fälle eintreten können, die in Tabelle 2 aufgelistet sind. Beispielsweise kann es sein, dass ein Label in den Trainingsdaten vergeben wurde, dass tatsächlich auch klassifiziert wurde (*true positive*). Ebenso können Labels für das betrachtete Element nicht vorhanden sein, die der Klassifikator ebenfalls nicht ausgewählt hat (*true negative*). Allerdings kann es auch passieren, dass ein Label nicht gefunden wurde, dass laut Trainingsdaten dem Element zugeschrieben wurde (*false negative*) oder dass ein Label fälschlicherweise zugeteilt wurde, dass das Element aber laut Trainingsdaten nicht beschreibt (*false positive*) (Manning, C. D., Raghavan, P., und Schütze, H. (2009): 155). Für die Evaluation wird zunächst gezählt, wie häufig diese Ereignisse in der Kreuzvalidierung vorkommen. Anschließend kann mittels verschiedener Evaluationsmaße festgestellt werden, wie effektiv das Klassifikationssystem tatsächlich ist.

[6] in der Klasse `DefaultExperimentGenerator`
[7] in der Klasse `SingleExperimenExecution`

	relevant	*nicht relevant*
gefunden	*true positives*	*false positives*
nicht gefunden	*false negatives*	*true negatives*

Tabelle 2: Zusammenhang zwischen gefundenen und relevanten Elementen

6.2. Evaluationsmaße

Da man Klassifikation unter anderem im Bereich des Information Retrievals einordnet, kann man die Evaluationskriterien, die dort verwendet werden, zunächst übernehmen. Häufig verwendete Maße, um Klassifikatoren zu evaluieren, sind deshalb *Precision*, *Recall* und das sogenannte *F*-Maß. Für Systeme im Information Retrieval sagt die *Precision* etwas darüber aus, wie viele der gefundenen Elemente auch tatsächlich für die Anfrage relevant sind. Sei D die getestete Datenmenge und H der zu evaluierende Klassifikationsalgorithmus, so gilt:

$$Precision(D,H) = \frac{true\ positives}{true\ positives + false\ positives}$$

Der *Recall* hingegen misst wie viele der insgesamt relevanten Elemente auch tatsächlich gefunden wurden:

$$Recall(D,H) = \frac{true\ positives}{true\ positives + false\ negatives}$$

Das *F*-Maß bildet das harmonische Mittel zwischen *Recall* und *Precision* und ist wie folgt definiert:

$$F(D,H) = \frac{(\beta^2 + 1) * Precision(D,H) * Recall(D,H)}{\beta^2 * Precision(D,H) + Recall(D,H)}$$

β stellt dabei eine reelle Zahl zwischen 0 und ∞ dar und dient dazu, *Precision* und *Recall* je nach Anwendung zu gewichten. Solange $0 < \beta < 1$ ist, betont das *F*-Maß die *Precision*, sollte $\beta > 1$ sein, so wird der *Recall* stärker gewichtet. In den meisten Anwendungen gilt $\beta = 1$, wodurch beide Evaluationsmaße gleich gewichtet werden (Manning, C. D., Raghavan, P., und Schütze, H. (2009): 156). Hier spricht man auch vom sogenannten *F1-Score*, der wie folgt definiert ist:

$$F_{\beta=1}(D,H) = \frac{2 * Precision(D,H) * Recall(D,H)}{Precision(D,H) + Recall(D,H)}$$

Bei dem Versuch, diese Evaluationsmaße bei einer Multilabel-Klassifikation anzuwenden, fällt auf, dass man zunächst definieren muss, ab wann ein Treffer beispielsweise als *true positive* gelten kann. So könnte man hier die Elemente zählen, bei denen alle vorhergesagten Labels dem Goldstandard entsprechen. Allerdings würde dann bereits ein Element als inkorrekt klassifiziert gelten, wenn ein Label zu viel oder zu wenig annotiert wurde. Es fehlt an dieser Stelle also an feineren Evaluationsmaßen, die auf Multilabel-Klassifikation abgestimmt sind. Deshalb werden nun unterschiedliche Maße vorgestellt, die Madjarov et al. für ihre Evaluation von Multilabel-Algorithmen nutzen (Madjarov, G. et al. (2012): 3089). Zunächst unterteilen Madjarov et al. die Evaluationsmaße in Zweiteilungs-basierte und Ranking-basierte Maße. Zweiteilung bezieht sich hier darauf, dass die Performanz des Algorithmus darauf untersucht wird, ob die vorhergesagten relevanten Labels mit den Labels in den Trainingsdaten übereinstimmen. Es wird also die binäre Relevanz aller Labels untersucht. Hier unterscheiden Madjarov et al. erneut zwischen Beispiel-basierten und Label-basierten Evaluationsmaßen.

6.2.1. Beispiel-basierte Evaluation

Beispiel-basierte Evaluationsmaße ermöglichen anhand der Trainingsdaten eine Aussage über die Unterschiede zwischen den Labels der Trainingsdaten und den klassifizierten Labels. Zu diesen Maßen gehören unter anderem der sogenannte *Hamming loss* sowie die *classification accuracy*. Das erste Maß ermittelt wie oft der binäre Wert des vorhergesagten Labels und der des Labels aus den Trainingsdaten voneinander abweichen. Dies ist wie folgt definiert:

$$hammingloss(h) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{Q}\,|h(x_i)\Delta Y_i|$$

Dabei stellt $h(x_i)$ das Set der klassifizierten Labels und Y_i das Set aus den Trainingsdaten dar. N bezeichnet die Anzahl der betrachteten Elemente und Q die Anzahl der möglichen Labels (Zhang, M.-L. und Zhou, Z.-H. (2007): 2039). Der *Hamming loss* sagt also aus wie hoch der Anteil der falsch gesetzten Labels pro Trainingselement durchschnittlich ist. Je näher der Wert also gegen 0 konvergiert desto besser ist die Beispiel-basierte Performanz des Algorithmus. Im Gegensatz zu *Precision* und *Recall* gibt der *Hamming loss* keinen Einblick darüber, in welche Richtung die Fehler entstanden sind, also ob es sich um *false positives* oder *false negatives* handelt.

Des Weiteren misst die *classification accuracy* wie groß der Anteil der Elemente ist, die vollständig richtig klassifiziert wurden. Sie wird folgendermaßen definiert:

$$classificationaccuracy(h) = \frac{1}{N}\sum_{i=1}^{N} I(Z_i = Y_i)$$

Dabei stellt Z_i die Menge der klassifizierten Labels dar. I nimmt bei einem wahren Ausdruck 1 und bei einem falschen Ausdruck 0 an, sodass schließlich die Summe der Elemente vorliegt, bei denen die vorhergesagte Menge und die richtige Menge vollständig übereinstimmen (Tsoumakas, G., Katakis, I., und Vlahavas, I. (2010): 679).

6.2.2. Label-basierte Evaluation

Label-basierte Maße evaluieren die durchschnittliche Effektivität des Algorithmus mithilfe der Effektivität jedes einzelnen Labels. Grundsätzlich wird dazu die Effektivität wie bei einer Singlelabel-Klassifikation über *Precision*, *Recall* und *F*-Maß berechnet. Dabei lassen sich Label-basierte Methoden in zwei Gruppen unterteilen (Tsoumakas, G., Katakis, I., und Vlahavas, I. (2010): 679). Die Makro-Mittelung berechnet für jedes Label l die Evaluationsergebnisse. Es wird also für l gezählt wie viele *true positives(tp)*, *false positives(fp)*, *true negatives(tn)* und *false negatives(fn)* es gibt. Diese Werte werden genutzt, um beispielsweise die *Precision* für l zu ermitteln. Anschließend wird über alle Evaluationsergebnisse der Labels das arithmetische Mittel gebildet. Formell ausgedrückt bedeutet dies für das Evaluationsmaß B:

$$B_{macro} = \frac{1}{q}\sum_{l=1}^{q} B(tp_l, fp_l, tn_l, fn_l)$$

Eine andere Label-basierte Evaluation ist die Mikro-Mittelung. Bei dieser werden alle *tp*, *fp*, *tn* und *fn* der Labels summiert und mithilfe der Summen das Evaluationsergebnis berechnet:

$$B_{micro} = B(\sum_{l=1}^{q} tp_l\,, \sum_{l=1}^{q} tf\,, \sum_{l=1}^{q} tn_l\,, \sum_{l=1}^{q} fn_l)$$

6.2.3. Ranking-basierte Evaluation

Neben den Maßen, die ermitteln wie gut die vorhergesagten Labels mit den Trainingsdaten übereinstimmen, evaluieren Ranking-basierte Evaluationsmaße, die ebenfalls in dieser Implementation Anwendung finden, einen Algorithmus im Hinblick darauf, wie gut das vorhergesagte Ranking zutrifft. Dieses beruht auf den tatsächlich berechneten Wahrscheinlichkeiten und kann mithilfe der Maße *one error* und *coverage* evaluiert werden. Ersteres ist für die Rankingfunktion f wie folgt definiert:

$$oneerror(f) = \frac{1}{N}\sum_{i=1}^{N}\left[\left[\left[\operatorname{argmax}_{l\in Y} f(x_i, l)\right] \notin Y_i\right]\right]$$

Es wird also der maximale Wert für f gesucht. Wenn das zugehörige Label l nicht in der vorgeschriebenen Labelmenge Y_i enthalten ist, nimmt der Ausdruck den Wert 1 an. Letztendlich wird aus der entstehenden Summe das arithmetische Mittel über alle Trainingsdaten gebildet. Somit gibt *one error* Aufschluss darüber, bei wie vielen Trainingsdaten das am wahrscheinlichsten vorhergesagte Label nicht im Goldstandard enthalten ist. *Coverage* evaluiert das Ranking in dem Sinne, dass geprüft wird wie viele Positionen man durchschnittlich im Rankingvektor abwärtsgehen muss, um alle Labels aus Y_i zu erreichen. Je näher sich dieser Wert an die Kardinalität der Trainingsdaten annähert desto besser ist das Ranking. $coverage(f)$ ist folgendermaßen definiert:

$$coverage(f) = \frac{1}{N}\sum_{i=1}^{N}\max_{l\in Y_i} rank_f(x_i, l) - 1$$

Dabei stellt $rank_f(x_i, l)$ den Rankingvektor für das Label l am Trainingselement $x_i$dar (Madjarov, G. et al. (2012): 3095f.).

6.3. Ergebnisse

Im Folgenden werden nun die Evaluationsergebnisse vorgestellt, die unter verschiedenen Konfigurationen erzielt wurden. Alle Ergebniswerte können im Anhang (Tabelle 6 und 7) eingesehen werden. Der fettgedruckte Wert ist dabei der Beste für das jeweilige Evaluationsmaß. In Bezug auf den *Hamming loss* erzielte der *ML-kNN*-Klassifikator mit *k*=16 das beste Ergebnis (0,0919). Dabei wurde die Distanz über die Cosinus-Ähnlichkeit und die Merkmalsgewichtung mithilfe der Log-Likelihood-Funktion berechnet. Für die Erstellung der Merkmale wurden die Normalisierung, der Füllwörter-Filter, der Stemmer sowie die Umwandlung in 3-Gramme verwendet. Bei der Setzung des Label-Sets wurden leere Sets erlaubt. Mit einem Mikro-*F*-Maß von 0,5907 befindet sich diese Konfiguration dort ebenfalls unter den besten Werten. Dabei ist die Mikro-*Precision* 0,7672 und der Mikro-*Recall* 0,4803. Das *coverage* liegt unter den angegebenen Konfigurationen bei 3,1353. Bei einer Label-Kardinalität von 1,66 bedeutet dies also, dass der Rankingvektor durchschnittlich 1,5 Labels mehr „durchlaufen“ werden muss, um alle Labels aus dem Goldstandard zu erreichen. Ebenfalls in Bezug auf das Ranking beträgt der Wert des *one error* 0,2378. Also sind knapp 24% der als am wahrscheinlichsten vorhergesagten Labels nicht im Goldstandard enthalten.

6.3.1. Makro-Mittelungen

Die Makro-*Precision* beträgt bei den genannten Konfigurationen 0,6179. Ein auffällig niedriger Makro-*Recall* von 0,2646 führt zu einem niedrigen Makro-*F*-Maß von 0,3309. Diese Werte lassen sich dadurch erklären, dass die Makro-Mittelungen über die Mittelung der Label-basierten Evaluationsergebnisse entstehen. Bei Betrachtung von *Precision*, *Recall* und *F*-Maß für die einzelnen Labels fällt auf, dass sich diese Werte stark voneinander unterscheiden (siehe Anhang: Tabelle 4). Am auffälligsten sind die Ergebnisse zu „Forschung & Entwicklung". Der Algorithmus hat dieses Label kein einziges Mal vergeben, wodurch alle Evaluationsmaße 0 ergeben. Dies könnte darauf zurückgeführt werden, dass nur sehr wenige Trainingsdaten (n = 58) mit diesem Label vorliegen. Dadurch ist es dem Algorithmus nicht möglich, von repräsentativen Beispielen zu lernen. Die am stärksten vertretenen Labels („Anwendungsentwicklung" und „Beratung & Consulting") erreichen mit 0,7864 und 0,6881 die besten *F*-Maße, was die Vermutung stützt, dass mehr Trainingsdaten zu besseren Ergebnissen führen. Auffällig ist auch, dass die *Precision* in 10 von 12 Fällen über 60% liegt. Neben dem bereits angesprochenen Label „Forschung & Entwicklung" ist die *Precision* von „Risk / Compliance Management" mit 0,4286 vergleichsweise niedrig. Allerdings stehen auch hier vergleichsweise wenige Trainingsdaten (n = 96) zur Verfügung.

6.3.2. Der Einfluss von *k*

Um den Einfluss von verschiedenen Werten von *k* zu ermitteln, werden alle übrigen Konfigurationen zunächst beibehalten. Dabei ergibt sich der *Hamming loss*, der in Abbildung 2 zu sehen ist. Hier wurde das *k* von 4 bis 16, jeweils in Dreierschritten, erhöht. Der *Hamming loss* verringert sich zwischen diesen Konfigurationen um 0,0065. Also werden bei *k*=16 im Schnitt 0,6% weniger Labels pro Stellenanzeige falsch gesetzt als bei *k*=4. Dieser Unterschied ist minimal, allerdings lässt sich an der Grafik ein starker Abfall zwischen *k*=4 und *k*=10 erkennen. Danach ändert sich der *Hamming loss* nicht mehr stark.

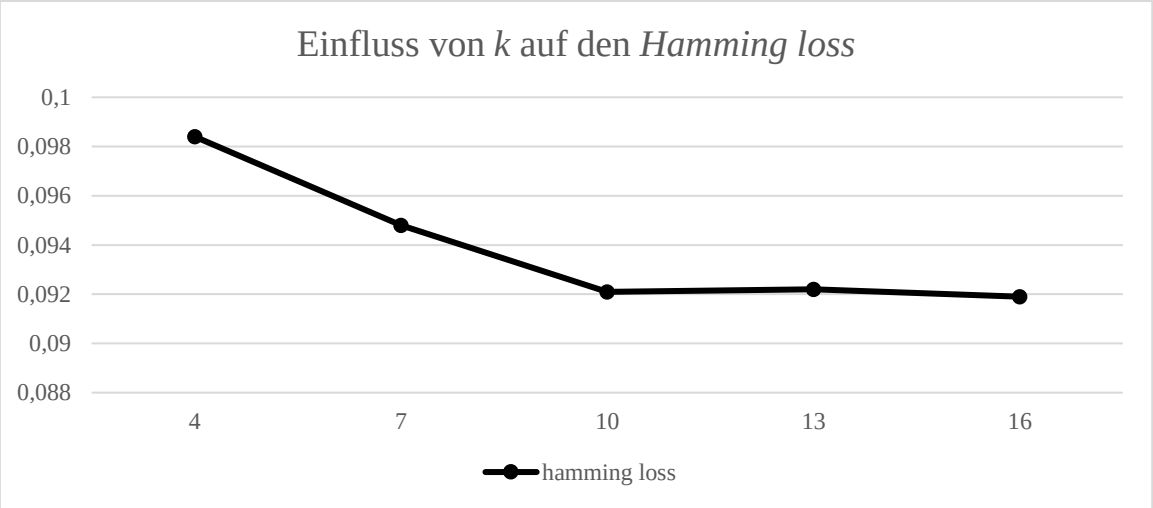

Abbildung 2(hamming loss)

Die Anzahl der nächsten Nachbarn hat ebenfalls einen Einfluss auf die *classification accuracy*, die Label-basierten sowie die Ranking-basierten Evaluationswerte (siehe Abbildung 3). In Bezug auf das Mikro-*F*-Maß lässt sich ein stetiger Anstieg von 4,2% bei steigendem *k* erkennen. *One error* verbessert sich ebenfalls stetig mit einem Verlust von knapp 3%. Die *classification accuracy* wird bei *k*=10 maximiert (0,3611). Auch der Mikro-*Recall* erzielt bei *k*=10 mit 48,1% sein bestes Ergebnis.

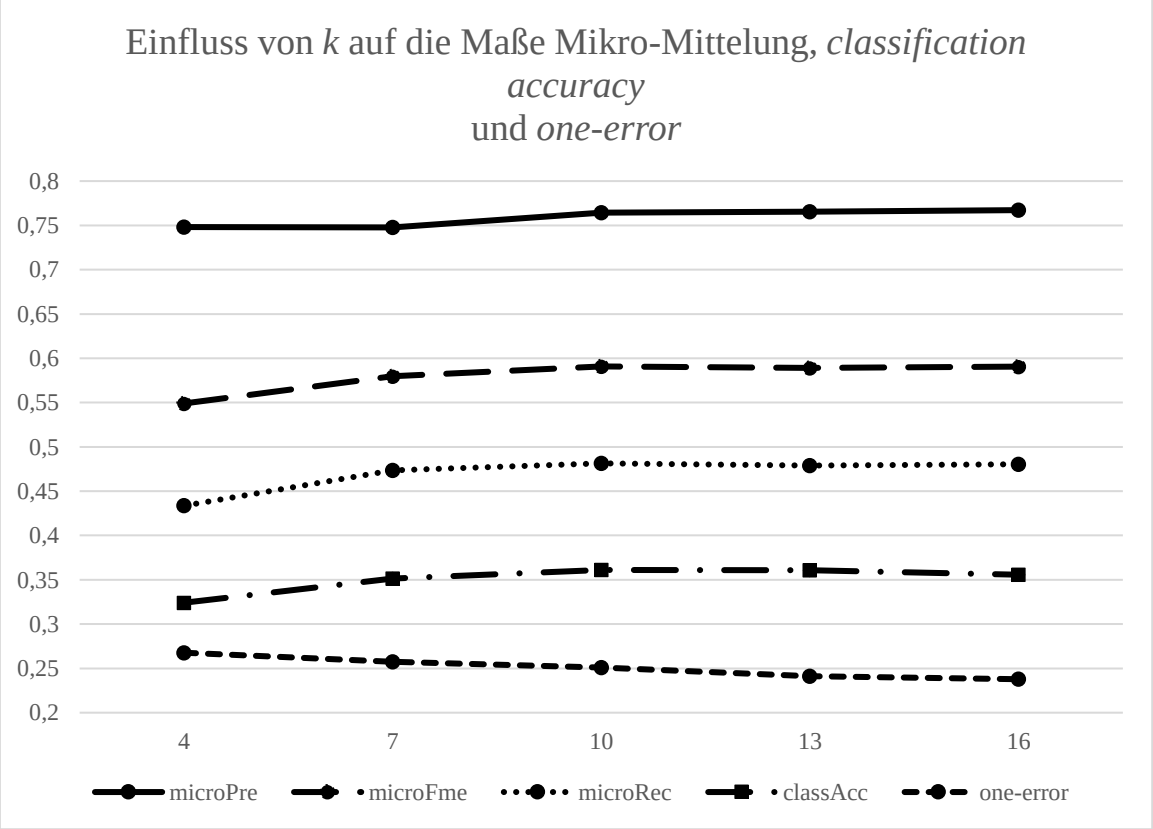

Abbildung 3(Mikro-Mittelung, classification accuracy, one-error)

6.3.3. Ergebnisse bei veränderten Konfigurationen

Sobald man keine leeren Label-Sets als Rückgabe zulässt, steigt der Mikro-*Recall* um 4% auf 0,5204. Allerdings verschlechtert sich gleichzeitig die Mikro-*Precision* um 3,6%. Daher ist das Ausfüllen von leeren Rückgaben nur bedingt sinnvoll. Einen ähnlichen Effekt erreicht man, wenn der Schwellwert für die Zuschreibung eines Labels von 0,5 auf 0,35 heruntergesetzt wird. Sowohl der *Recall* als auch das *F*-Maß steigen in Makro- und Mikro-Mittelung an, allerdings leidet darunter jeweils die *Precision* und auch der *Hamming loss* steigt um 0,4%. Bei der Ausgabe der klassifizierten Labels ist es also die Entscheidung des Anwenders, ob genaue Labels oder viele Labels wichtiger sind. *Naive Bayes* erzielt bei Default-Konfigurationen wesentlich schlechtere Ergebnisse. Der *Hamming loss* liegt hier bei 0,3048, das Mikro-*F*-Maß bei 0,2298. Die entsteht vor allem durch eine sehr ungenaue Mikro-Präzision von 0,1765.

Der Verzicht auf N-Gramme führt überwiegend zu geringfügig schlechteren Ergebnissen. Eine Ausnahme bildet die Mikro-*Precision* mit einem Anstieg von 0,3%. Außerdem führt die Erzeugung von Merkmalen ohne N-Gramme zu einer Feature-Anzahl von 19214 (Default=8316), was die Rechenzeit erheblich verlängert. Sowohl die Art der Gewichtung der Merkmale als auch die Art der Distanzberechnung haben keine große Auswirkung auf die Ergebnisse der Klassifikation. Die Ermittlung der Ähnlichkeit zwischen zwei Merkmalsvektoren über die euklidische Distanz lässt die Ergebnisse der Evaluation im Hinblick auf die Makro-Mittelung, *one-error*, *coverage* sowie Mikro-*Recall* leicht nach unten abweichen. Im Gegensatz dazu verbessern sich die *classification accuracy* (+1,1%) und die Mikro-*Precision* (0,4%) leicht. Die Gewichtung der Terme über das Tf-idf-Maß hat ähnliche Auswirkungen. Beispielsweise verbessert sich der Mikro-*Recall* um 0,5%, während sich die Mikro-*Precision* um 1,1% verschlechtert.

7. Diskussion

Da die Evaluationsergebnisse stark von der Beschaffenheit der zu klassifizierenden Daten abhängen, ist es schwierig direkte Vergleiche zu anderen Studien, die mit *ML-kNN* gearbeitet haben, herzustellen. Zhang und Zhou testen ihren Algorithmus beispielsweise an der Klassifikation von Web Pages (Zhang, M.-L. und Zhou, Z.-H. (2007): 2046). Dabei sind 14 Top-Level-Kategorien in Second-Level-Kategorien unterteilt, die klassifiziert werden sollen. Als Trainingsdaten lagen für den Versuch auf jeder Top-Level-Ebene 2000 Trainingsdaten sowie 3000 Testdaten vor. Für die Top-Level-Kategorien erreicht *ML-kNN* einen durchschnittlichen *Hamming loss* von 0,0426. Damit ist *ML-kNN* bei der Klassifikation von Web Pages in Bezug auf den *Hamming loss* minimal besser als der in Kapitel 3.2 erwähnte Rank-SVM (0,0434). Die Klassifikation von Stellenausschreibungen fällt mit 0,0922 schlechter aus, allerdings bestehen

die Merkmalsvektoren bei der Klassifikation der Web Pages nur aus durchschnittlich 655 Merkmalen. Da die Vektoren der Stellenausschreibungen bei den als beste Konfigurationen deklarierten Einstellungen 8316 Einträge umfassen, könnte eine stärkere Merkmalsreduktion die Implementation verbessern. Dazu käme der Einsatz eines *Mutual Information*-Filters in Frage. Dieser prüft, welche Merkmale große Auswirkungen für die Entscheidung für ein Label haben. In diesem Gebiet gibt es bereits Ansätze, die speziell für Multilabel-Klassifikationen entwickelt werden (Lee, J. und Kim, D.-W. (2013)).

Ebenso könnte eine gleichmäßigere Verteilung der Labels zu besseren Ergebnissen führen. Diese Möglichkeit ist allerdings relativ unrealistisch, da die Trainingsdaten aus dem tatsächlichen Arbeitsmarkt stammen und so für manche Schwerpunkte nur wenige Stellen angeboten werden. Auch kann es sein, dass die Labels manuell inkonsistent vergeben wurde, da keine Prüfung der Validität der Trainingsdaten möglich ist. Um die ungleichmäßige Verteilung und Kombination der Labels aufzuwiegen, wäre es eine Möglichkeit, Label-Abhängigkeiten genauer zu untersuchen. Beispielsweise könnte es sein, dass es sich für manche der Kombinationen lohnen würde, ein neues Label zu eröffnen, da diese Gruppe von Stellenanzeigen einen eigenen Charakter aufweist.

Eine weitere Möglichkeit, um die Klassifikation der Stellenausschreibungen zu verbessern, wäre die Einbindung von externem Wissen. Ein Versuch in diese Richtung wurde bereits über eine Schlagwort-Klassifikation unternommen. Dabei wurden die Stellenausschreibungen auf Schlagwörter hin geprüft, die ebenfalls zur manuellen Klassifikation verwendet wurden. Allerdings führte dies zu einigen *false positives*, was darauf schließen lässt, dass die Schlagwörter nicht Schwerpunkt-spezifisch genug sind.

8. Fazit

Die Klassifikation von Stellenausschreibungen kann mit der vorliegenden Implementation insofern durchgeführt werden, als dass für jede unbekannte Stellenanzeige ein Ranking aller Labels erstellt wird, an dem man sich bei der manuellen Annotation orientieren kann. Da es sich um einen Algorithmus aus dem Bereich des maschinellen Lernens handelt, können durch eine größere Menge an Trainingsdaten bessere Ergebnisse erzielt werden. Da Klassifikatoren aus dem Machine Learning als flexibler als regelbasierte Klassifikatoren gelten, ist es möglich, neue Labels hinzuzufügen, solange nur genügend Trainingsdaten vorhanden sind. Der Einsatz einer Problemtransformation in Kombination mit *Naive Bayes* führte zu keinen zufriedenstellenden Ergebnissen. Die Implementation des *ML-kNN*, einer Algorithmus-Adaption des *kNN*, lieferte

Ergebnisse, auf denen aufgebaut werden kann. Herkömmliche Evaluationsmaße wie *Precision*, *Recall* und *F*-Maß sind für Multilabel-Klassifikatoren nur bedingt aussagekräftig, sodass andere Evaluationsmaße herangezogen wurden, die aus der Literatur zur Multilabel-Klassifikation stammen.

9. Literaturnachweise

Bensberg, F., und Buscher, G.
2016 Job Mining als Analyseinstrument für das Human-Resource-Management. *HMD Praxis der Wirtschaftsinformatik* 53: 815–27. https://doi.org/10.1365/s40702-016-0256-3 (accessed 1 September 2017).

Carstensen, K.-U.; Ebert, C.; Endriss, C.; Jekat, S.; Klabunde, R.; und Langer, H., eds.
2004 *Computerlinguistik und Sprachtechnologie: Eine Einführung*. 2nd ed. München: Elsevier.

Chen, X.-j.; Zhan, Y.-z.; Ke, J.; und Chen, X.-b.
2016 Complex video event detection via pairwise fusion of trajectory and multi-label hypergraphs. *Multimedia Tools and Applications* 75: 15079–100. https://doi.org/10.1007/s11042-015-2514-8.

Dunning, T.
1993 Accurate Methods for the Statistics of Surprise and Coincidence. *Computational Linguistics* 19: 61–74.

Elisseeff, A., Weston, J.
2002 A kernel method for multi-labelled classification. Pp. 681–87 in *Advances in neural information processing systems 14: Proceedings of the 2002 conference*, ed. T. G. Dietterich. Cambridge, Mass.: MIT Press.

Feldman, R., und Sanger, J.
2008 *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge: Cambridge Univ. Press.

Hermes, J., und Schandock, M.
2016 *Stellenanzeigenanalyse in der Qualifikationsentwicklungsforschung: Die Nutzung maschineller Lernverfahren zur Klassifikation von Textabschnitten*. Bonn: Bundesinstitut für Berufsbildung.

Herrera, F.; Charte, F.; Rivera, A. J.; und del Jesus, M. J.
2016 *Multilabel Classification: Problem Analysis, Metrics and Techniques*. Cham, s.l.: Springer International Publishing.

Jurafsky, D., und Martin, J. H.
2009 *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd ed. Prentice Hall series in artificial intelligence. Upper Saddle River, NJ: Prentice Hall Pearson Education Internat.

Lee, J., und Kim, D.-W.
2013 Feature selection for multi-label classification using multivariate mutual information. *Pattern Recognition Letters* 34: 349–57. https://doi.org/10.1016/j.patrec.2012.10.005.

Madjarov, G.; Kocev, D.; Gjorgjevikj, D.; und Džeroski, S.
2012 An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition* 45: 3084–3104. https://doi.org/10.1016/j.patcog.2012.03.004.

Manning, C. D.; Raghavan, P.; und Schütze, H.
2009 *Introduction to Information Retrieval*. Cambridge: Cambridge Univ. Press.

Mikheev, A.
2005 Text Segmentation. Pp. 201–18 in *The Oxford handbook of computational linguistics*, ed. R. Mitkov. Oxford: Oxford Univ. Press.

Mooney, R. J.
2005 Machine Learning. Pp. 376–94 in *The Oxford handbook of computational linguistics*, ed. R. Mitkov. Oxford: Oxford Univ. Press.

Russell, S. J.
1995 *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Englewood Cliffs, N.J: Prentice Hall.

Sebastiani, F.
2002 Machine learning in automated text categorization. *ACM Computing Surveys* 34: 1–47. https://doi.org/10.1145/505282.505283.

Spyromitros E.; Tsoumakas G.; und Vlahavas I.
2008 An Empirical Study of Lazy Multilabel Classification Algorithms. Pp. 401–6 in *Artificial intelligence: theories, models and applications: 5th Hellenic Conference on AI, SETN 2008, Syros, Greece, October 2 - 4, 2008 ; proceedings*, ed. J. Darzentas, G. A. Vouros, S. Vosinakis, und A. Arnellos. Lecture notes in computer science Lecture notes in artificial intelligence 5138. Berlin: Springer.

Tsoumakas, G., und Katakis, I.
2008 Multi-Label Classification: An Overview. Pp. 64–74 in *Data warehousing and mining: Concepts, methodologies, tools, and applications*, ed. J. Wang. Premier reference source. Hershey, Pa.: Information Science Reference.

Tsoumakas, G.; Katakis, I.; und Vlahavas, I.
2010 Mining Multi-label Data. Pp. 667–85 in *Data Mining and Knowledge Discovery Handbook*, ed. O. Maimon und L. Rokach. Boston, MA: Springer US.

Vens, C.; Struyf, J.; Schietgat, L.; Džeroski, S.; und Blockeel, H.
2008 Decision trees for hierarchical multi-label classification. *Machine Learning* 73: 185–214. https://doi.org/10.1007/s10994-008-5077-3.

Zampieri, M.; Hermes, J.; und Schwiebert, S.
2013 Identification of Patterns and Document Ranking of Internet Texts: A Frequency-based Approach. Pp. 25–40 in *Non-standard Data Sources in Corpus-based Research*, ed. M. Zampieri und S. Diwersy. 1st ed. ZMS-Studien - Schriften des Zentrums Sprachenvielfalt und Mehrsprachigkeit der Universität zu Köln Vol. 5. Aachen: Shaker.

Zhang, M.-L., und Zhou, Z.-H.
2007 ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition* 40: 2038–48. https://doi.org/10.1016/j.patcog.2006.12.019.

9.1. Internetquellen

Get in GmbH, ULR: https://www.get-in-it.de/

Apache POI-XSSF: https://poi.apache.org/spreadsheet/

jsoup: https://jsoup.org/

Die beschriebene Implementation befindet sich auf der beiliegenden CD sowie auf Github unter:

https://github.com/johannabi/JobAd_IE

Alle in dieser Arbeit angesprochenen ausführbaren Klassen befinden sich im package `de.uni_koeln.spinfo.ml_classification_applications`

10. Anhang

Tabelle 3(Häufigkeit der Schwerpunkte)

Schwerpunkt	Häufigkeit in den Trainingsdaten
Administration	561
Anwendungsentwicklung	1828
Beratung / Consulting	1141
Business Analysis	273
Datenbankentwicklung / BI	432
Forschung & Entwicklung	58
Produktmanagement	95
Projektmanagement	424
Quality Assurance	335
Risk / Compliance Management	96
System Engineering	275
Webentwicklung	537

Tabelle 4(Evaluationsergebnisse der einzelnen Labels)

Schwerpunkt	Recall	Precision	F-Maß
Administration	0,5615	0,8098	0,6632
Anwendungsentwicklung	0,7811	0,7919	0,7864
Beratung & Consulting	0,6281	0,7609	0,6881
Business Analysis	0,0441	0,6667	0,0828
Datenbankentwicklung / BI	0,2407	0,6582	0,3525
Forschung & Entwicklung	0,0	0,0	0,0
Produktmanagement	0,0211	0,6667	0,0408
Projektmanagement	0,1182	0,6494	0,2
Quality Assurance	0,2567	0,789	0,3874
Risk / Compliance Management	0,125	0,4286	0,1935
System Engineering	0,1091	0,6667	0,1875
Webentwicklung	0,2701	0,6682	0,3846

Tabelle 5(Konfigurationen für die Klassifikation)

Attribut	Wert
inputFile	Pfad zur .xlsx-Datei, in der die zu klassifizierenden Stellenanzeigen stehen
outputFolder	Pfad zum Ordner, in den die .xlsx-Datei mit den klassifizierten Dateien gelegt werden soll
focusesFile	Pfad zur .xlsx-Datei, in der die Schwerpunkte definiert sind
output	„ranking“: Ausgabe aller Schwerpunkte mit tatsächlichen Wahrscheinlichkeiten; „labels“: Liste mit Schwerpunkten, deren Wahrscheinlichkeiten über dem Schwellwert liegen
ignoreStopwords	„true“: Füllwörter werden nicht in den Merkmalsvektor aufgenommen; „false“: Merkmalsvektor wird mit Füllwörtern gebildet
normalizeInput	„true“: Tokens werden für den Merkmalsvektor normalisiert; „false“: Tokens werden nicht normalisiert
useStemmer	„true“: Tokens werden für den Merkmalsvektor gestemmt; „false“: Tokens werden nicht gestemmt
nGrams	Länge(n) der N-Gramme (als Ganzzahl, mit Kommata getrennt) (optional)
distance	„cosinus“, „euklid“ oder „manhattan“ als Distanzmaß im Vektorraum (nur, falls es sich um *ML-kNN* handelt)
threshold	Schwellwert, ab welcher Wahrscheinlichkeit ein Label gesetzt werden soll (als Double; min.: 0.0, max.: 1.0) (optional)
knnValue	Anzahl der betrachteten nächsten Nachbarn (als Ganzzahl) (nur, falls es sich um *ML-kNN* handelt)
allowemptylabelmap	„true“: falls bei einer Stellenanzeige keine Label-Wahrscheinlichkeit den Schwellwert überschreitet, soll kein Label gesetzt werden; „false“: in o.g. Fall soll das wahrscheinlichste Label gesetzt werden
classifier	„mlknn“ oder „naivebayes“ als Klassifikator
quantifier	„tfidf“ oder „loglikelihood“ als Art der Vektorgewichtung

Tabelle 6(Evaluationsergebnisse für Mikro- und Makro-Mittelungen)

Makro-Precision	Makro-Recall	Makro-F-Maß	Mikro-Precision	Mikro-Recall	Mikro-F-Maß	Veränderung zum Default
0,5652	0,2314	0,294	0,748	0,4336	0,549	*k* = 4
0,7023	0,2564	0,322	0,7477	0,4735	0,5798	*k* = 7
0,6391	0,2711	0,3409	0,7644	0,4814	0,5908	*k* = 10
0,6297	0,263	0,3306	0,7655	0,4791	0,5893	*k* = 13
0,6179	0,2646	0,3309	0,7672	0,4803	0,5907	--
0,6021	0,2933	0,3535	0,731	0,5204	0,608	Leere Label-Sets: false
0,5618	0,3389	**0,3847**	0,6818	**0,5695**	**0,6206**	Schwellwert: 0.35
0,5668	0,2603	0,3225	**0,7714**	0,4773	0,5897	Distanz: Euklid
0,5763	0,2527	0,3157	0,7706	0,4669	0,5815	N-Gramme: null
0,5829	0,2992	0,3667	0,7562	0,486	0,5918	Gewichtung: Tf-idf
0,2442	**0,3595**	0,2187	0,1765	0,3292	0,2298	Algorithmus: Naive Bayes

Tabelle 7(Evaluationsergebnisse für Hamming loss, one-error und coverage)

Classification Accuracy	Hamming Loss	One-error	Coverage	Veränderung zum Default
0,3241	0,0984	0,2677	3,3455	*k* = 4
0,3515	0,0948	0,2575	3,2348	*k* = 7
0,3611	0,0921	0,251	3,1888	*k* = 10
0,3608	0,0922	0,2411	3,1636	*k* = 13
0,3559	**0,0919**	**0,2378**	3,1356	--
0,3882	0,0927	0,2378	3,1351	Leere Label-Sets: false
0,3441	0,0962	0,2378	3,1351	Schwellwert: 0.35
0,3671	0,0918	0,2419	3,1962	Distanz: Euklid
0,3529	0,0929	0,2537	3,2214	N-Gramme: null
0,3553	0,0926	0,2414	**3,0929**	Gewichtung: Tf-idf
0,0066	0,3048	0,7562	7,6844	Algorithmus: Naive Bayes

Default-Konfigurationen:
Algorithmus: *ML-kNN*, *k*=16, leere Label-Sets erlauben: *true*, Schwellwert: 0.5, Distanz: Cosinus, Gewichtung: Log-Likelihood, N-Gramme: 3-Gramme, Füllwörter-Filter: *true*, Normalisierung: *true*, Stemmer: *true*

Hiermit versichere ich an Eides Statt, dass ich diese Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Stellen meiner Arbeit, die dem Wortlaut und dem Sinn nach anderen Werken und Quellen, einschließlich der Quellen aus dem Internet, entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen, Karten und Abbildungen.

Diese Arbeit habe ich in gleicher oder ähnlicher Form oder auszugsweise nicht im Rahmen einer anderen Prüfung eingereicht.

Ich versichere zudem, dass der Text der eingereichten elektronischen Fassung mit dem Text der vorgelegten Druckfassung identisch ist.

Köln, _____ Unterschrift: _____