technische universität
dortmund

Bachelorarbeit

Implementierung von Konfidenz-basierten Aktivenlern-Methoden in RapidMiner

Tim Schendekehl
Januar 2015

Gutachter:
Prof. Dr. Katharina Morik
Dipl.-Inform. Christian Pölitz

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (VIII)
http://ls8-www.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Wenn große Datenmengen automatisch mit maschinellem Lernen klassifiziert werden sollen, dann müssen zuerst einige Beispiele von Hand annotiert werden. Im KobRA-Projekt [1] werden z. B. Texte anhand von linguistischen Merkmalen klassifiziert. Dazu muss zuerst ein Experte für jedes Beispiel angeben, zu welcher Klasse es gehört. Dieser Vorgang wird Annotieren genannt. Das Annotieren ist aber recht aufwendig und verursacht Kosten, wie z. B. Arbeitszeit.

Um den Aufwand zu verringern ist es sinnvoll, wenn diese Beispiele gezielt ausgewählt werden. Eine Möglichkeit dazu ist aktives Lernen [23]. Hierbei hat der Computer die Möglichkeit Fragen an einen Experten zu stellen, anstatt nur eine Menge annotierter Daten zu bekommen. Dazu kann der Computer z. B. aus einer Menge nicht annotierter Daten auszuwählen, welche Beispiele annotiert werden sollen, und kann so den Informationsgewinn der Beispiele erhöhen.

Für aktives Lernen gibt es verschiedene Möglichkeiten. Eine ist, mit einem bereits vorhandenen Modell zu berechnen, wie sicher sich das Modell für neue Daten ist und nur unsichere Beispiele auszuwählen.

In dieser Bachelorarbeit wird ein Plugin für den RapidMiner [19] entwickelt, mit dem die Klassifizierung von Texten durch aktives Lernen unterstützt wird. Das Lernen selbst findet über die bereits im RapidMiner integrierte libSVM [9] und das Text Processing Plugin statt.

Für das aktive Lernen werden neue Operatoren benötigt, die neue Beispiele auswählen und durch den Benutzer annotieren lassen. Das Ganze findet in einer Schleife statt, die auch durch einen Operator repräsentiert wird. Abbildung 1.1 zeigt, wie die Daten in der Schleife verarbeitet werden.

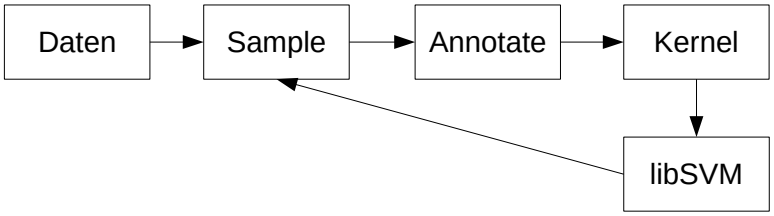

Abbildung 1.1: In einer Schleife werden neue Beispiele ausgewählt und zum Lernen verwendet.

Bei den Daten handelt es sich um Texte, die klassifiziert werden sollen. Der Sample-Schritt wählt aus den Texten die Beispiele aus, die vom Benutzer annotiert werden sollen. Dieser Schritt wird im RapidMiner später durch einen Operator dargestellt.

Im ersten Durchlauf werden die Beispiele zufällig ausgewählt. Danach wird für jedes Beispiel mit den bereits annotierten Beispielen die Konfidenz berechnet. Jetzt werden nur noch Beispiele mit geringer Konfidenz ausgewählt. Außerdem soll es möglich sein, die neuen Beispiele immer zufällig auszuwählen. So kann überprüft werden, ob und wie gut das aktive Lernen die Ergebnisse verbessert.

Nachdem die Beispiele ausgewählt wurden, annotiert der Benutzer die Beispiele im Annotate-Schritt. Wenn nur das aktive Lernen getestet wird und die Daten bereits annotiert sind, dann entfällt dieser Schritt. Danach wird mit den neuen und alten Daten zusammen ein neues Modell gelernt. Das gelernte Modell kann im nächsten Durchlauf von dem Sample-Schritt verwendet werden.

1.2 Aufbau der Arbeit

In Kapitel 2 wird zunächst das Problem der Textklassifikation und die Lösung mit maschinellem Lernen erklärt. Wie man die Kosten beim Annotieren mit aktiven Lernen reduziert, wird dann in Kapitel 3 erklärt. Hier wird neben dem implementierten Verfahren auch auf Alternativen eingegangen. Danach wird in Kapitel 4 die Implementierung beschrieben. In Kapitel 5 werden dann die durchgeführten Experimente beschrieben und die Ergebnisse verglichen. Zum Schluss gibt es in Kapitel 6 eine Folgerung. Zusätzlich ist in Anhang A eine Beschreibung zur Benutzung des Plugins.

Kapitel 2

Textklassifikation

Bei der Textklassifikation geht es darum, automatisch zu überprüfen, ob für einen Text bestimmte Eigenschaften gelten. Dazu wird die Menge der möglichen Texte in mehrere Klassen unterteilt, wobei die Klassen zu den Eigenschaften gehören, die man überprüfen möchte. Beispiele für solche Eigenschaften sind, ob der Text einer Bewertung in einem Internetportal positiv oder negativ ist.

Ein Beispiel aus dem KobRA-Projekt [1] ist das Klassifizieren nach linguistischen Merkmalen. Hier wird die Verwendung von Sprache an großen Mengen von Texten, die auch Korpora genannt werden, untersucht.

Im einfachsten Fall gibt es genau zwei Klassen, die z. B. mit positiv (1) und negativ (-1) bezeichnet werden. Es können aber auch mehr Klassen sein.

Optimal wäre es, wenn man direkt einen Algorithmus entwirft, der die Klasse zu einem Text genau berechnet. Das ist allerdings nicht immer möglich oder wäre sehr aufwendig. Es könnte z. B. sein, dass Menschen zwar intuitiv entscheiden können, zu welcher Klasse ein Text gehört, aber nicht allgemein sagen können, nach welchem Verfahren sie vorgehen.

Um das Problem trotzdem zu lösen, wird im maschinellen Lernen eine Menge von Beispielen benutzt, mit der ein Klassifikator gelernt wird, der auch andere Texte klassifizieren können soll. Ein Beispiel besteht hier aus dem Text und kann mit der Klasse zu der der Text gehört versehen sein. Zum Lernen eines Klassifikators gibt es mehrere Möglichkeiten. In dieser Arbeit wird eine SVM benutzt, was in Abschnitt 2.1 erklärt wird.

Jetzt stellt sich die Frage, woher die Beispiele kommen. Manchmal liegen bereits viele Beispiele vor, die man benutzen kann. Wenn Benutzer z. B. in einem Internetportal Bewertungen schreiben und direkt angeben, ob ihre Bewertung positiv oder negativ ist, dann kann man die Daten direkt als Beispiele benutzen. Es kann aber auch sein, dass man zwar schon viele Texte hat, aber noch nicht die dazu gehörenden Klassen kennt. Hier müssen die Daten also erst mit den Klassen markiert werden, was auch Annotieren genannt wird.

Das Annotieren kann allerdings recht aufwendig sein und verursacht Kosten. Eventuell kann auch nur ein Experte die Klassen ermitteln. Deshalb ist die Zahl der Beispiele even-

tuell sehr beschränkt. Im KobRA-Projekt müssen die Texte z. B. erst manuell von einem Linguisten annotiert werden.

2.1 SVM

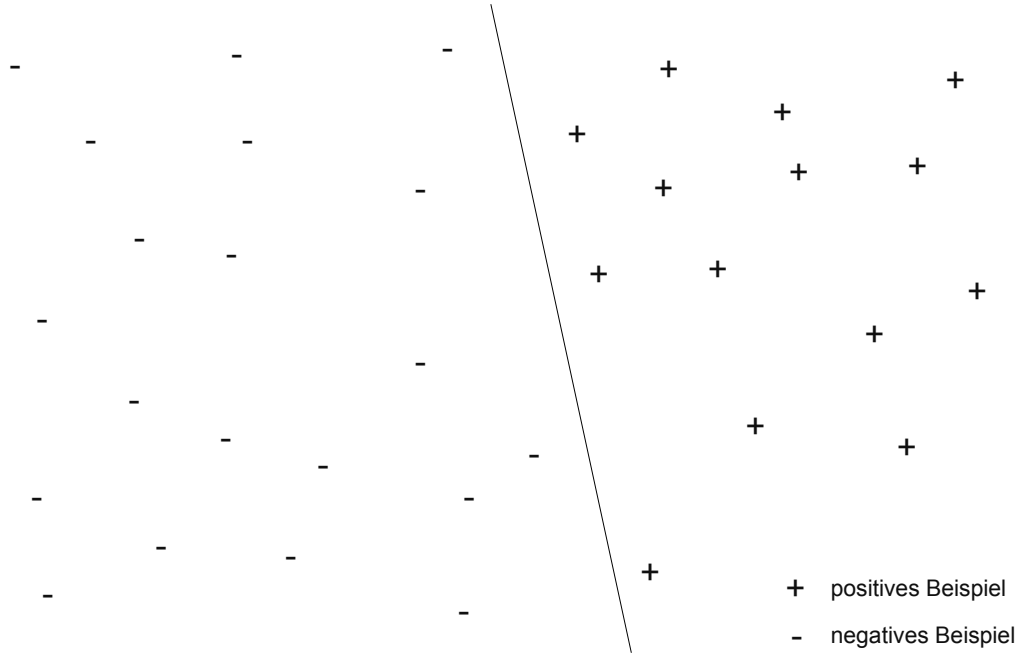

Abbildung 2.1: Beispiele aus zwei Klassen werden durch eine Ebene genau getrennt

Eine Möglichkeit zur automatischen Klassifikation von Daten ist die Support Vector Machine [8].

Die Beispiele werden dabei als Vektoren in einem euklidischen Vektorraum $\mathbb{R}^d$ aufgefasst. In dem Fall, dass es nur zwei Klassen gibt, versucht die SVM eine Hyperebene zu finden, sodass alle positiven Beispiele auf der einen und alle negativen Beispiele auf der anderen Seite liegen. Wenn das möglich ist, dann gibt es in der Regel mehrere Möglichkeiten, wie die Hyperebene gelegt werden kann. Man möchte hier, dass der Klassifikator auf andere Daten gut generalisiert, also dass auch andere Daten später gut klassifiziert werden. Deshalb wird die Hyperebene so gelegt, dass der Abstand zu den Beispielen maximiert wird. Es ergibt sich hier also ein Optimierungsproblem, das man lösen muss, um die Hyperebene zu erhalten.

Wenn die Beispiele $\left\{(x_i, y_i) \mid i = 1, \ldots, l,\, y_i \in \{-1, 1\},\, x_i \in \mathbb{R}^d\right\}$ gegeben sind, dann wird eine Hyperebene gesucht, die durch $w \in \mathbb{R}^d$ und $b \in \mathbb{R}$ gegeben ist. Die Punkte $x \in \mathbb{R}^d$ mit $w \cdot x + b = 0$ liegen dann genau auf der Hyperebene. Das Vorzeichen von

$w \cdot x + b$ wird als Vorhersage für ein Beispiel x benutzt. Um eine Hyperebene zu finden, die die Beispiele trennt wird $\frac{1}{2}\|w\|^2$ minimiert, wobei $\forall i : y_i (x_i \cdot w + b) - 1 \geq 0$ gelten muss.

Es lässt sich aber nicht immer eine solche Hyperebene finden. Manchmal liegen die Daten nicht so, dass eine Hyperebene sie trennen kann. Es kann auch sein, dass es Beispiele gibt, die falsch annotiert wurden oder seltene Ausnahmen darstellen. Um trotzdem eine Hyperebene zu finden kann man solche Ausreißer erlauben. Dazu wird das Optimierungsproblem so angepasst, dass Ausreißer vermieden werden müssen. Da es hier jetzt verschiedene Möglichkeiten gibt, wie stark Ausreißer bestraft werden, gibt es den Parameter C. Wenn C hoch ist, dann werden Ausreißer stärker bestraft und die Abstände der Hyperebene zu den Beispielen ist weniger wichtig. Bei einem niedrigeren C werden stärkere Ausreißer in Kauf genommen, um die Abstände zu vergrößern.

Dazu werden die Schlupfvariablen ξ_i für $i = 1, \ldots, l$ eingeführt. Jetzt wird $\frac{1}{2}\|w\|^2 - C \sum_i \xi_i$ minimiert, wobei $\forall i : y_i (x_i \cdot w + b) - 1 + \xi_i \geq 0$ und $\forall i : \xi_i \geq 0$ gelten müssen.

Beim Klassifizieren von neuen Daten wird überprüft, auf welcher Seite der Hyperebene das Beispiel liegt, also welches Vorzeichen $w \cdot x + b$ hat. Der Abstand des Beispiels von der Hyperebene kann auch benutzt werden, um zu berechnen, wie unsicher ein Beispiel ist. Es gibt auch verschiedene Möglichkeiten für ein Beispiel x die Wahrscheinlichkeit $P_\Theta (y \mid x)$ abzuschätzen, dass es zu Klasse y gehört. Die libSVM [9] benutzt dazu die zweite Methode aus [28].

Die Hyperebene, die durch das Optimierungsproblem ermittelt wurde, hängt nicht von allen Vektoren ab. Nur die Vektoren, die der Hyperebene am nächsten sind, sind für die genaue Position der Hyperebene wichtig. Diese Vektoren werden Stützvektoren genannt.

2.1.1 Kernel

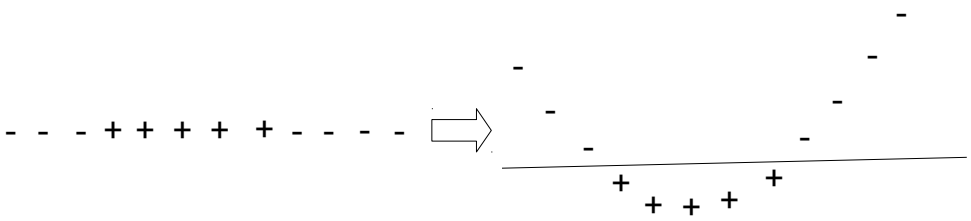

Abbildung 2.2: Die Beispiele auf der linken Seite liegen in einem 1-dimensionalen Raum und lassen sich so nicht linear trennen. Durch Transformation in den 2-dimensionalen Raum rechts, lässt sich eine Trenngerade finden.

Bis jetzt wurde davon ausgegangen, dass die SVM jedes Beispiel als Vektor bekommt. Die Daten liegen allerdings als Texte vor. Außerdem besteht noch das Problem, dass sich

die Daten eventuell nicht sinnvoll durch eine Hyperebene trennen lassen. Um das zu lösen, werden Kernel benutzt, die die Daten in einen höher dimensionalen Raum abbilden.

Wenn $\mathcal{H}$ ein höher dimensionaler Hilbertraum ist und $\Phi : \mathbb{R}^d \mapsto \mathcal{H}$ eine Funktion die Beispiele in diesen Raum abbildet, dann kann man das Optimierungsproblem auch in dem höher dimensionalen Raum lösen. Das duale Problem zu dem vorherigen Optimierungsproblem ist [8, S. 14]:

$$\begin{aligned} Maximize \quad & L_D \equiv \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j \\ subject\,to \quad & 0 \leq \alpha_i \leq C \\ & \sum_i \alpha_i y_i = 0 \end{aligned}$$

Hier sieht man, dass die Vektoren der Beispiele nur in den Skalarprodukten $y_i y_j$ und $x_i x_j$ vorkommen. Im höher dimensionalen müsste man deshalb auch nur $\Phi(x_i)\,\Phi(x_j)$ berechnen. Diese Berechnung kann man durch $K(x_i, x_j)$ ersetzen für eine Funktion $K : X \times X \to \mathbb{R}$. Man muss die Funktion Φ also gar nicht berechnen, sondern kann stattdessen immer K berechnen. Die Menge X muss deshalb auch nicht mehr $\mathbb{R}^d$ sein, sondern kann auch z. B. Texte beinhalten. Die Funktion K wird auch Kernel-Funktion genannt und berechnet, wie ähnlich sich zwei Beispiele sind.

Eine einfache Kernel-Funktion ist das Skalarprodukt. Die gaußsche radiale Basisfunktion (kurz RBF) mit $K(x, y) = e^{-\|x-y\|^2/2\sigma^2}$ [8, S. 21] ist eine andere Kernel-Funktion, die auch für Daten, die sich nicht linear trennen lassen, gute Ergebnisse erzielt. Hier gibt es den weiteren Parameter σ, der zusätzlich zu C festgelegt werden muss.

Eine Möglichkeit die Texte auf Vektoren abzubilden ist der Bag of Words. Hierbei wird der Text zunächst in Wörter aufgeteilt, indem der Text bei bestimmten Zeichen, wie Leerzeichen oder Satzzeichen getrennt wird. Der Text wird dann auf einen Vektor abgebildet, der für jedes Wort, das in dem Text vorkommen könnte die Anzahl der tatsächlichen Vorkommen enthält. Die Vektoren können dann noch normalisiert werden. Eine Kernel-Funktion könnte dann das Skalarprodukt der Wortvektoren berechnen.

Die Wortvektoren müssen aber nicht linear trennbar sein. Benutzt man den RBF-Kernel können die Beispiele eventuell besser getrennt werden. Der Bag of Words reicht aber nicht immer aus. Wenn sich Beispiele nur durch die Reihenfolge der Wörter unterscheiden, dann würden sie als gleich betrachtet werden. Manchmal muss man also andere Kernel benutzen.

Eine Möglichkeit ist, alle Teilstrings des Textes als Merkmale zu benutzen. Der String Subsequence Kernel [13] betrachtet hierzu alle Teilstrings mit einer bestimmten Länge und gewichtet sie danach, wie weit sie über den Text verteilt sind.

Es ist auch möglich den Text zuerst zu Parsen, also in einen Baum umzuwandeln, der die Syntax des Textes wiedergibt. In [17] wird beschrieben, wie man die Bäume für einen Kernel benutzen kann.

Kapitel 3

Aktives Lernen

Wenn das Annotieren eines Beispiels Kosten verursacht, weil z. B. ein Experte die Beispiele von Hand annotieren muss, dann wäre es schön, wenn nur möglichst wenige Beispiele annotiert werden müssen, um einen geeigneten Klassifikator zu erhalten. Eine Möglichkeit dazu ist aktives Lernen. Hierbei wird dem Computer erlaubt, Fragen an den Benutzer zu stellen. Eine solche Frage ist, zu welcher Klasse ein Beispiel gehört. Die einfachste Möglichkeit ist die Beispiele nur zufällig auszuwählen. Mit aktivem Lernen gibt es aber verschiedene Verfahren, die Beispiele intelligenter auszuwählen.

In [23] werden dazu drei Szenarien unterscheiden:

1. membership query synthesis
2. stream-based selective sampling
3. pool-based sampling

Bei membership query synthesis werden die Beispiele komplett neu generiert. Bei Texten würde das bedeuten, dass alle möglichen Zeichenfolgen generiert werden können und so viele Texte annotiert werden müssen, die später gar nicht relevant sind.

Bei dem Szenario stream-based selective sampling wird für eine Folge von Beispielen jedes mal entschieden, ob das Beispiel annotiert werden soll.

In dieser Arbeit wird das Szenario pool-based sampling verwendet. Hierbei liegt bereits eine Menge unklassifizierter Beispiele vor, aus denen zu annotierende Beispiele ausgewählt werden können.

Für die Wahl der zu annotierenden Beispiele gibt es auch wieder mehrere Verfahren. Eine Möglichkeit ist, wenn man schon ein Modell hat, mit diesem Modell zu bestimmen, wie sicher es sich für die einzelnen Beispiele ist. Gewählt werden dann die Beispiele, bei denen sich das Modell am unsichersten ist. Dieses Verfahren wird *uncertainty sampling* genannt. Wenn zum Lernen des Modells eine SVM benutzt wird, dann kann die berechnete Konfidenz benutzt werden. Da zu Beginn noch kein Modell vorliegt, müssen zunächst zufällige Beispiele gewählt werden (siehe Abschnitt 3.1.1).

Es ist zwar möglich immer nur ein neues Beispiel auszuwählen und ein neues Modell zu lernen. Allerdings kann das zu aufwendig sein. Deshalb können jeweils mehrere neue Beispiele ausgewählt werden.

3.1 Uncertainty Sampling

$\mathcal{U}$ = Menge der nicht annotierten Beispiele
$\mathcal{L}$ = Menge der zu Beginn annotierten Beispiele
while $|\mathcal{U}| > 0 \wedge$ Abbruchkriterium nicht erfüllt **do**
  $\Theta = \text{train}(\mathcal{L})$
  wähle $X \subseteq \mathcal{U}$ mit $|X| = \min(n, |\mathcal{U}|)$, sodass die Beispiele in X nach Modell Θ möglichst unsicher sind
  for all $x \in X$ **do**
    Frage nach dem Label y für x bei dem Experten
    $\mathcal{L} \leftarrow \mathcal{L} \cup (x, y)$
  end for
  $\mathcal{U} \leftarrow \mathcal{U} \setminus X$
end while

Algorithmus 3.1: Algorithmus für aktives Lernen mit Konfidenz. Pro Runde werden n Beispiele ausgewählt.

Die Idee von uncertainty sampling ist, mit einem bereits gelernten Modell für alle zur Verfügung stehenden Beispiele zu berechnen, wie sicher sich das Modell für eine Klasse entscheiden kann. Es werden dann die Beispiele ausgewählt, bei denen das Modell sich am unsichersten ist. Für die Definition der Unsicherheit gibt es verschiedene Möglichkeiten. Algorithmus 3.1 zeigt das grundsätzliche Vorgehen. Gestartet wird mit einer Menge nicht annotierter Beispiele $\mathcal{U}$ und einer bereits annotierten Menge Beispiele $\mathcal{L}$. Die Menge $\mathcal{L}$ kann zufällig aus $\mathcal{U}$ gewählt und annotiert werden. Danach wird eine Schleife solange durchlaufen, wie $\mathcal{U}$ nicht leer ist und mögliche zusätzliche Abbruchkriterien nicht erfüllt sind. In jeder Runde wird ein Modell Θ gelernt und benutzt, um eine Menge X zu wählen, die annotiert und zu $\mathcal{L}$ hinzugefügt wird.

Von der SVM für Modell Θ erhält man für ein Beispiel x die Wahrscheinlichkeit $P_\Theta(y \mid x)$, dass Beispiel x das Label y hat. Es gibt jetzt mehrere Möglichkeiten, wie aus diesen Wahrscheinlichkeiten berechnet wird, wie unsicher ein Beispiel ist. In [22, S. 13 ff.] werden drei verschiedene Maße beschrieben.

Bei *Least Confident* wird das Beispiel ausgewählt, bei dem die vorhergesagte Klasse die geringste Konfidenz hat. Die vorhergesagte Klasse für ein Beispiel x ist $\hat{y}$ und das ausgewählte Beispiel ist dann x^*_{LC}:

$$\begin{aligned} \hat{y} &= \arg\max_y P_\Theta(y \mid x) \\ x^*_{LC} &= \arg\min_x P_\Theta(\hat{y} \mid x) \end{aligned}$$

Hier wird allerdings nur eine Klasse berücksichtigt und die Informationen der anderen Klassen gehen verloren.

Mit dem Maß *Margin* wird der Abstand der Konfidenzen der zwei wahrscheinlichsten Klassen minimiert. Das ausgewählte Beispiel ist dann x^*_M, wobei $\hat{y}_1$ das wahrscheinlichste und $\hat{y}_2$ das zweit wahrscheinlichste Label ist.

$$x^*_M = \arg\min_x \left[P_\Theta(\hat{y}_1 \mid x) - P_\Theta(\hat{y}_2 \mid x)\right]$$

Das Maß *Entropy* betrachtet die Konfidenzen aller Klassen und maximiert die Entropie. Es wird dann x^*_H ausgewählt.

$$x^*_H = \arg\max_x -\sum_y P_\Theta(y \mid x) \log P_\Theta(y \mid x)$$

Alle drei Maße sind für nur zwei Klassen äquivalent. In der Implementierung des Plugins in RapidMiner wurde *Least Confident* benutzt, da in den Experimenten nur zwei Klassen benutzt werden.

In [24] werden zwei verschiedene Arten von Unsicherheit unterschieden. Es kann sein, dass sich ein Modell unsicher über ein Beispiel ist, weil es starke Hinweise gibt, die in beide Richtungen deuten (most-surely uncertain) oder weil es gar nicht genug Daten hat (least-surely uncertain). Indem beim Auswählen Beispiele bevorzugt wurden, die most-surely uncertain waren konnten in [24] Verbesserungen gegenüber normalem uncertainty sampling gemacht werden.

3.1.1 Startmenge

Zu Beginn des Lernprozesses stehen noch keine annotierten Daten zur Verfügung, mit denen ein Modell gelernt werden könnte. Die neu zu annotierenden Beispiele können also noch nicht anhand der Konfidenz ausgewählt werden. Eine Möglichkeit ist sie zufällig zu bestimmen.

Wichtig ist hierbei, dass genug Beispiele ausgewählt werden, sodass alle möglichen Klassen vorhanden sind. Ansonsten könnte kein sinnvolles Modell gelernt werden.

In [18, S. 23 f.] werden noch andere Methoden zum Bestimmen der Startmenge vorgestellt. Eine Möglichkeit [15] besteht darin, in den Daten nach besonders dicht bei einander liegenden Beispielen zu suchen, und die in den Zentren dieser Gebiete liegenden Beispiele zu wählen.

In dieser Arbeit wurde nur eine zufällige Startmenge genommen. Um zu verhindern, dass die zufällige Startmenge die Ergebnisse zu stark beeinflusst, wurden die Experimente mehrfach ausgeführt.

3.1.2 Annotieren mehrerer Beispiele pro Runde

Um dafür zu sorgen, dass jedes Beispiel, das annotiert wird, maximalen Informationsgewinn liefert, wäre es am besten jedes Beispiel einzelnes auszuwählen und annotieren zu lassen. Das kann allerdings zu aufwendig sein, um das Benutzen von aktivem Lernen zu rechtfertigen. Deshalb ist es sinnvoll, in jeder Runde mehrere Beispiele zu wählen und annotieren zu lassen.

Hierbei entsteht allerdings das Problem, dass sich die besten Beispiele recht ähnlich sein können und deshalb nicht alle Beispiele tatsächlich notwendig sind. Eine Möglichkeit das zu lösen ist, die Ähnlichkeit der neuen Beispiele untereinander zu berücksichtigen und keine Beispiele auszuwählen, die zu ähnlich sind. Ein konkreter Algorithmus hierzu wird in [7] beschrieben.

Weitere Methoden, um mehrere Beispiele gleichzeitig auszuwählen, werden in [23, S. 35] und [18, S. 25 f.] beschrieben. In dieser Arbeit wurde nur das Wählen auf Basis der Konfidenz implementiert.

3.1.3 Parameterwahl für die SVM

Die SVM hat verschiedene Parameter, die geeignet gewählt werden müssen, um das beste Ergebnis zu erzielen. Der Parameter C zur Gewichtung von Ausreißern ist hier immer vorhanden. Je nach benutztem Kernel gibt es noch weitere Parameter.

Eine Möglichkeit, geeignete Werte zu finden, ist mehrere Wertekombinationen auszuprobieren und mit einer Kreuzvalidierung zu überprüfen, wie gut die Werte sind. Diese Methode wird auch Gridsearch genannt. Bei der Kreuzvalidierung wird die Menge der annotierten Daten in mehrere Teilmengen aufgeteilt und jede Teilmenge wird einmal als Testmenge für das Modell, das auf allen anderen Teilmengen gelernt wird, benutzt. Mit dem Durchschnitt der Genauigkeiten kann man dann Abschätzen, wie gut das Modell auf allen Beispielen ist.

Wenn die Beispiele allerdings durch aktives Lernen ausgewählt werden, ergibt sich hier ein Problem. Bei der Kreuzvalidierung kann das Ergebnis jetzt verfälscht werden, weil die Testmengen auch Beispiele enthalten, die mit aktivem Lernen ausgewählt wurden. In [2] wird beschrieben, wie man dieses Problem lösen kann, indem zwei getrennte Mengen von annotierten Beispielen verwaltet werden. Eine Menge enthält die zum Lernen verwendeten Beispiele und wird mit aktivem Lernen gefüllt. Die andere Menge enthält nur zufällig ausgewählte Beispiele und wird zum Testen der Modelle verwendet. Dabei muss immer wieder überprüft werden, bei welcher Menge sich neue Beispiele mehr lohnen.

Ein anderes Problem bei Gridsearch ist, dass die Laufzeit erhöht wird. Wenn die Annotation durch einen Benutzer vorgenommen wird, kann das zu langen Unterbrechungen führen, in denen der Benutzer warten muss.

3.1.4 Stoppkriterium

Da es beim aktiven Lernen darum geht, die Kosten, die durch das Annotieren entstehen, zu minimieren, stellt sich die Frage, wie viele Beispiele man annotieren sollte. Wenn die maximalen Kosten vorgegeben sind und man das beste Modell möchte, kann man einfach die Zahl der Beispiele festlegen. Allerdings kann es sein, dass man irgendwann genug Beispiele annotiert hat, sodass sich das Modell durch weitere Beispiele nicht mehr verbessern lässt. In dem Fall könnte man die Kosten senken, indem man keine weiteren Beispiele annotiert.

Hierzu muss man erkennen, wann sich das Modell nicht mehr stark verbessern lässt. Eine Möglichkeit das abzuschätzen ist, die minimale Konfidenz der noch nicht annotierten Daten zu betrachten [29, 4.1 Maximum Uncertainty Method]. Ist sie über einem bestimmten Wert, dann ist sich der Klassifikator bei allen nicht annotierten Daten schon so sicher, dass man davon ausgehen kann, dass diese Beispiele das Modell nicht mehr verbessern werden. Deshalb kann man in diesem Fall vorzeitig das aktive Lernen abbrechen und so Kosten sparen.

3.2 Kosten

Aktives Lernen wird benutzt, um die Kosten, die beim Annotieren entstehen, zu minimieren. Dazu muss zunächst definiert werden, was die Kosten sind. Eine einfache Definition ist die Kosten als die Anzahl an annotierten Beispielen zu nehmen und diese Definition wird auch in dieser Arbeit benutzt.

Es sind aber auch andere Definitionen möglich [23, S. 37 ff.] [18, S. 31 ff.]. Zum Beispiel kann die Zeit, die ein Benutzer für die Annotation braucht als Kosten genommen werden. In der Praxis kann es auch vorkommen, dass die Kosten von dem Benutzer abhängen, der die Annotation vornimmt.

Da die tatsächlichen Kosten erst nach der Annotation bekannt sind, müssen die Kosten approximiert werden, um beim aktiven Lernen berücksichtigt zu werden. Zum Beispiel kann die Länge der Texte benutzt werden, um die Kosten abzuschätzen.

3.3 Alternativen und Verbesserungsmöglichkeiten

In dieser Arbeit wurde nur Uncertainty Sampling mit Konfidenz implementiert. Es gibt aber verschiedene Probleme, die dabei auftreten können. Außerdem gibt es noch andere Arten von aktivem Lernen.

3.3.1 Unterrepräsentierte Klassen

In der Praxis kann es vorkommen, dass die Klassen nicht gleich stark vertreten sind [22, S. 72 f.]. Es ist z. B. möglich, dass die positive Annotation im Vergleich zur negativen Annotation sehr selten ist. Hier kann es schwierig sein, überhaupt genug positive Beispiele zu finden. Da zum Lernen des ersten Modells von allen Klassen Repräsentanten vorhanden sein müssen, muss in diesem Fall die Startmenge recht groß gewählt werden. Bei dem zufälligen Auswählen der Beispiele werden von der häufigeren Klasse auch immer mehr gewählt.

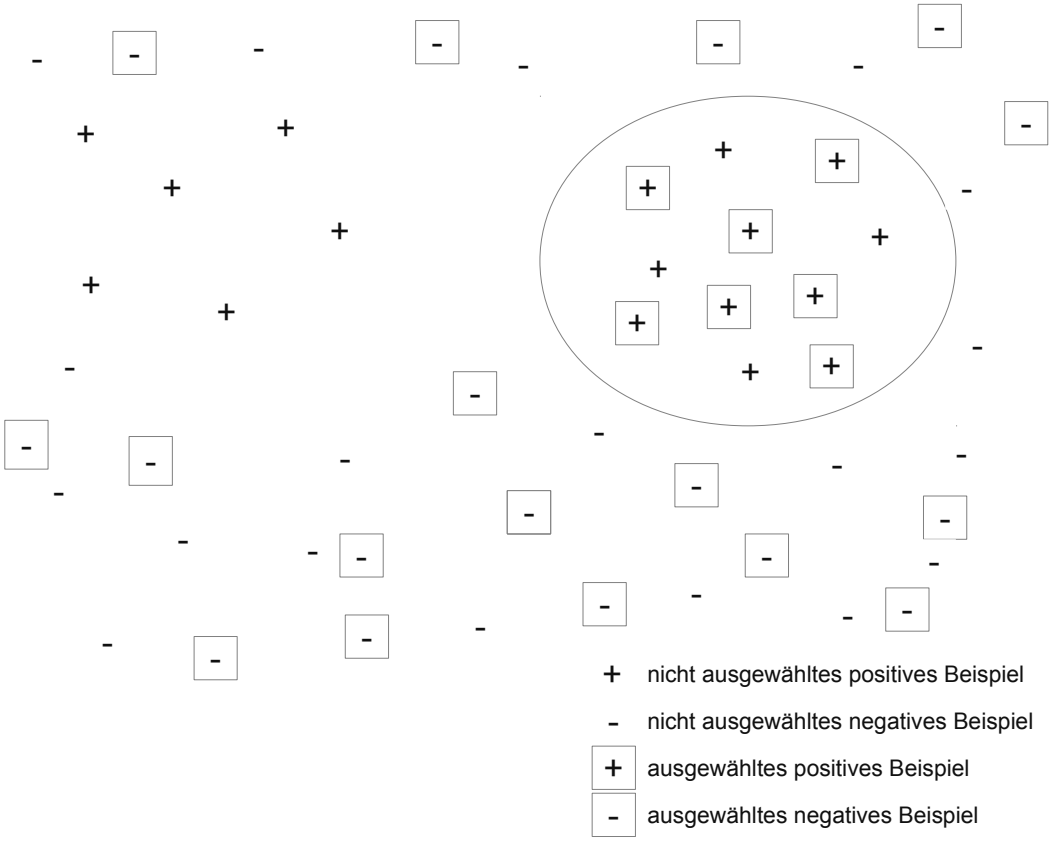

Abbildung 3.1: Ein Bereich von positiven Beispielen ist so weit vom Rand des Modells entfernt, dass die Beispiele durch aktives Lernen erst spät ausgewählt werden.

Wenn aktives Lernen benutzt wird, dann werden zwar Beispiele ausgewählt, die in unsicheren Bereichen liegen, allerdings kann das auch dazu führen, dass manche Beispiele der selteneren Klasse gar nicht gefunden werden. In Abbildung 3.1 sieht man ein Beispiel, in dem es sehr viele negative Beispiele gibt, aber weniger positive Beispiele. Die positiven Beispiele verteilen sich dabei auf zwei Bereiche, wobei nur in dem rechten Bereich schon positive Beispiele ausgewählt wurden. Das gelernte Modell, das durch die Ellipse dargestellt wird, betrachtet deshalb auch nur den rechten Bereich als positiv. Die positiven Beispiele

des linken Bereichs sind allerdings so weit von dem Rand des Modells entfernt, dass sie bei aktivem Lernen nicht oder erst spät ausgewählt werden. Wenn sich die Beispiele auf noch mehr Bereiche verteilen, dann kann sich der Effekt noch verstärken.

In [3] wird untersucht, ob man bei ungleich stark vertretenen Klassen schneller ein gutes Modell lernen kann, wenn der Benutzer nicht nur Beispiele annotiert, sondern auch direkt nach Beispielen der seltenen Klasse sucht. Dabei wurde festgestellt, dass man auf diese Weise bei sehr ungleich verteilten Klassen die Genauigkeit deutlich schneller erhöhen kann, als mit aktivem Lernen oder zufälligem Auswählen der Beispiele. Kombiniert man das mit aktivem Lernen, können die Ergebnisse weiter verbessert werden.

3.3.2 Berücksichtigen der Verteilung der Daten

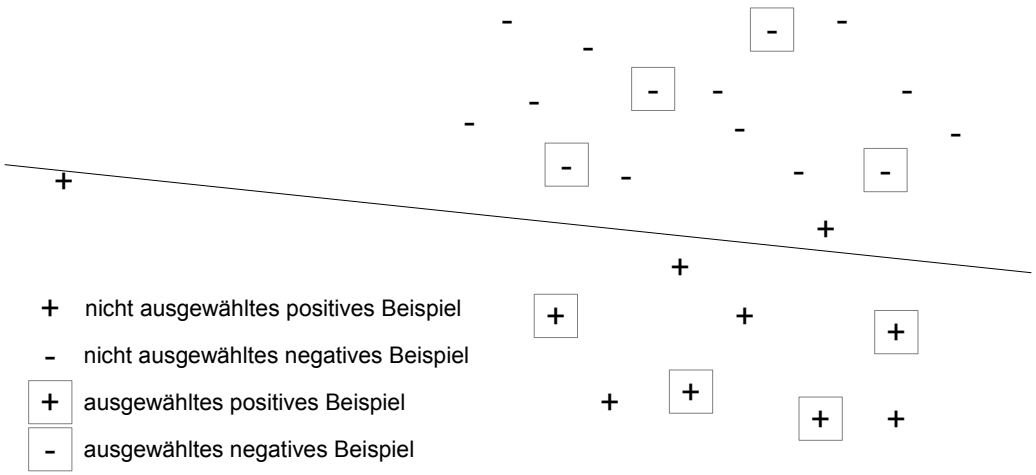

Abbildung 3.2: Zwar gibt es auf der linken Seite ein Beispiel das am nächsten an der Trenngeraden liegt, aber dieses Beispiel würde nur für wenige Beispiele einen Informationsgewinn bringen.

Wenn man nur für jedes Beispiel einzeln entscheidet, wie gut es zum Lernen ist, dann kann es passieren, dass Beispiele ausgewählt werden, die nur Ausreißer sind oder nur wenig Informationsgewinn für andere Beispiele bringen. Häufig sind die Beispiele nicht gleichmäßig verteilt. Es gibt Bereiche mit sehr vielen Beispielen und Bereiche mit wenig oder keinen Beispielen. Wenn man davon ausgehen kann, dass die Stichprobe eine ähnliche Verteilung hat, wie die echten Daten, die später klassifiziert werden sollen, dann kann man die Verteilung bei aktivem Lernen nutzen [22, S. 47 ff.]. Beispiele die in Bereichen mit hoher Dichte liegen, sollten bevorzugt werden, da später in diesem Bereich häufiger Beispiele klassifiziert werden müssen, während Beispiele in Bereichen mit geringer Dichte für die später zu klassifizierenden Daten nur wenig Relevanz haben. In Abbildung 3.2 sieht man ein Beispiel

hierzu. Auf der linken Seite ist ein einzelnes noch nicht ausgewähltes Beispiel in der Nähe der Trenngeraden und auf der rechten Seite sind viele Beispiele sehr dicht beieinander. Betrachtet man nur die Konfidenz der Beispiele würde man das linke Beispiel auswählen. Die Beispiele auf der rechten Seite könnten das Modell aber so verbessern, dass es später mehr Beispiele richtig klassifiziert.

In [11] werden verschiedene Methoden zu aktivem Lernen beschrieben und danach unterteilt, ob sie Korrelationen zwischen den Beispielen berücksichtigen. Manche der Methoden teilen die Daten z. B. erst in Cluster ein, um bessere Beispiele auswählen zu können.

3.3.3 Der Versionsraum

Aktives Lernen lässt sich mit binärer Suche vergleichen. Binäre Suche ist ein Verfahren, mit dem man schnell in einem Array den Index findet, ab dem eine bestimmte Aussage gilt, wobei angenommen wird, dass es einen Index gibt ab dem die Aussage immer gilt und davor nie. Anstatt alle Elemente des Arrays durchzugehen, bis man den Index gefunden hat, wird der Bereich, in dem der Index sich befinden kann immer weiter geteilt, bis er gefunden wurde. So kann die Laufzeit von $O(n)$ auf $O(log(n))$ verringert werden.

Das Prinzip der binären Suche lässt sich auf Klassifizierungsprobleme verallgemeinern [22, S. 21 ff.]. Eine Hypothese ist ein Modell, das für jedes mögliche Beispiel im Merkmalsraum $\mathcal{F}$ eine Vorhersage macht, zu welcher Klasse es gehört. Wenn eine SVM benutzt wird, wäre eine Hypothese also eine Hyperebene. Der Hypothesenraum $\mathcal{H}$ ist dann die Menge aller Hypothesen. Wenn man bereits Beispiele mit Annotation hat, dann sind nicht alle Hypothesen möglich. Der Versionsraum [16] $\mathcal{V} \subseteq \mathcal{H}$ ist die Menge aller Hypothesen, die mit den Beispielen konsistent sind. Es kann passieren, dass es gar keine Hypothese gibt, die mit allen Beispielen konsistent ist, dann wäre der Versionsraum leer.

Das Ziel von aktivem Lernen ist, mit möglichst wenig Beispielen ein möglichst genaues Modell zu erhalten. Geht man analog zu binärer Suche vor, dann versucht man den Versionsraum mit jedem neuen Beispiel möglichst stark zu verkleinern. Am besten ist also ein Beispiel, das mit am meisten Hypothesen im Versionsraum inkonsistent ist. Da man bei der Auswahl des Beispiels aber noch nicht die Klasse des Beispiels kennt, ist auch nicht bekannt, welchen Hypothesen es widerspricht. Um bei allen möglichen Annotationen des Beispiels möglichst viele Hypothesen zu verwerfen, sollte das Beispiel für alle Klassen gleich viele Hypothesen verwerfen. Bei binärer Suche wird das erreicht, indem ein Index getestet wird, der in der Mitte der möglichen Indizes liegt.

Auch wenn man mit der Konfidenz der SVM ein neues Beispiel auswählt, wird versucht den Versionsraum zu verkleinern. Zwar werden nicht alle möglichen Modelle betrachtet, um das beste neue Beispiel zu finden, aber das aktuelle Modell der SVM wurde so gewählt, dass die Hyperebene einen möglichst großen Abstand zu den Beispielen auf beiden Seiten hat. Durch Verschieben der Hyperebene könnte man auf beiden Seiten Hypothesen erzeugen,

die auch im Versionsraum liegen, wenn es bei den Beispielen keine Ausreißer gibt. Ein Beispiel, das genau auf der Hyperebene liegt, würde also alle Hypothesen verwerfen, die durch Verschieben der Hyperebene in eine Richtung entstehen.

In [26] wird ein Verfahren beschrieben, bei dem mit einer SVM der Versionsraum möglichst stark verkleinert wird. Dabei wird ausgenutzt, dass es eine Dualität zwischen dem Hypothesenraum $\mathcal{H}$ und dem Merkmalsraum $\mathcal{F}$ gibt. Hyperebenen in $\mathcal{F}$ sind Punkte in $\mathcal{H}$ und Punkte in $\mathcal{F}$ sind Hyperebenen in $\mathcal{H}$, denn die Beispiele trennen die möglichen Hypothesen, von den mit diesem Beispiel inkonsistenten Hyperebenen. Wenn man eine SVM im Merkmalsraum $\mathcal{F}$ anwendet, erhält man eine Hyperebene, die die Beispiele mit größtem Abstand trennt. Übersetzt in den dualen Hypothesenraum $\mathcal{H}$ bedeutet das, man erhält den Mittelpunkt der Kugel mit maximalem Radius m, die in dem Versionsraum liegt.

Da man versuchen will den Versionsraum möglichst schnell zu verkleinern, sucht man ein Beispiel, das den Versionsraum am besten halbiert. Dazu werden in [26] mehrere Möglichkeiten vorgeschlagen. Mit *Simple Margin* wird ein Beispiel gesucht, das im Hypothesenraum $\mathcal{H}$ zu einer Hyperebene gehört, die möglichst nah am Mittelpunkt der vorher erwähnten Kugel liegt. Das funktioniert aber nur gut, wenn der Versionsraum recht symmetrisch ist und die Kugel einen Großteil des Versionsraumes ausfüllt. Die Idee der Methode *MaxMin Margin* ist, dass der Radius der Kugel eine Approximation der Größe des Versionsraumes ist. Um abzuschätzen, wie gut ein Beispiel den Versionsraum trennt, wird für beide möglichen Klassen des Beispiels eine neue SVM gelernt, und die Oberfläche der Kugel im Versionsraum berechnet. Die beiden Radien werden mit m^+ und m^- und die Oberflächen mit $Area(\mathcal{V}^+)$ und $Area(\mathcal{V}^-)$ bezeichnet. Wenn $Area(\mathcal{V}^+)$ und $Area(\mathcal{V}^-)$ gleich groß sind, dann trennt das Beispiel den Versionsraum am besten. Deshalb wird das Beispiel gewählt, bei dem $\min(Area(\mathcal{V}^+), Area(\mathcal{V}^-))$ maximal ist. Es kann allerdings passieren, dass der Versionsraum sehr lang ist und deshalb m^+ und m^- beide sehr klein sind. Bei der Methode *Ratio Margin* wird deshalb die relative Größe der Radien betrachtet und man versucht $\min\left(\frac{m^-}{m^+}, \frac{m^+}{m^-}\right)$ zu maximieren.

3.3.4 Selektive Auswahl

Anstatt aus einer Menge von Beispielen das beste auszuwählen kann man bei einem Strom von Beispielen jeweils entscheiden ob ein Beispiel annotiert oder verworfen werden soll. Diese Strategie wird stream-based selective sampling genannt.

Eine Möglichkeit, die auf der Idee den Versionsraum zu verkleinern basiert, ist query by disagreement [10] [22, S. 24 ff.]. Dabei wird eine Menge gültiger Hypothesen gespeichert. Für jedes neue Beispiel wird überprüft, ob sich alle aktuellen Hypothesen einig über die Klasse dieses Beispieles sind. Wenn alle Hypothesen das gleiche vorhersagen, dann wird Beispiel weggeworfen. Ansonsten wird die Klasse bei dem Experten nachgefragt und die widersprechenden Hypothesen werden entfernt. Allerdings kann es sein, dass sich nicht alle

Hypothesen speichern lassen. Um das zu lösen kann man versuchen nur die speziellste und die allgemeinste Hypothese zu speichern.

Eine weitere Möglichkeit ist query by committee [22, S. 28 ff.] [18, S. 6 ff.]. Dabei gibt es immer eine Menge von Klassifizierern, die für jedes neue Beispiel eine Vorhersage machen. Wenn sich die Klassifizierer stark widersprechen, muss das Beispiel annotiert werden. Query by committee kann auch bei pool-based sampling benutzt werden, indem das Beispiel bei dem sich die Klassifizierer am stärksten widersprechen gewählt wird.

Um sinnvolle Ergebnisse zu erhalten werden verschiedene Klassifikatoren benötigt, die sich auch widersprechen. Mit den Methoden Bagging und Boosting [14] kann man aus den annotierten Daten mehrere unterschiedliche Klassifikatoren gewinnen. Bei Bagging [6] werden dazu unterschiedliche Teilmengen der annotierten Beispiele zum Lernen benutzt. Mit jeder Teilmenge wird ein anderer Klassifikator gelernt. Bei Boosting beeinflussen sich die Klassifikatoren gegenseitig.

Ein Algorithmus, der auch mit teilweise fehlerhaften Annotationen klarkommt ist A^2 oder Agnostic Active Learning [4]. Hierbei werden aus einem Strom die Beispiele ausgewählt, die den Versionsraum möglichst halbieren.

3.3.5 Auswahl der besten Strategie

Da es verschiedene Strategien gibt, wie man bei aktivem Lernen neue Beispiele auswählt, muss man sich vorher entscheiden, welche Strategie man wählt. Es ist vorher aber noch nicht klar, welche Strategie die beste sein wird. In [12] wird beschrieben, wie automatisch verschiedene Möglichkeiten ausprobiert werden können, um sich automatisch für die besten Strategien zu entscheiden.

Dazu wurde in [12] ein Algorithmus namens Active Learning by Learning entwickelt. Hierbei wurde eine Analogie zu dem multi-armed bandit problem benutzt. Ein zusätzliches Problem ist, dass die verschiedenen Strategien zu unterschiedlichen Zeiten unterschiedlich gut sein können. Außerdem kann es sein, dass mehrere Strategien das gleiche Beispiel auswählen und so Informationen über mehrere Strategien gewonnen werden.

3.3.6 Unzuverlässige Annotation

Es kann sein, dass nicht alle Annotationen der Daten richtig sind. Der Experte kann z. B. einen Fehler machen oder die Klasse ist für ein Beispiel gar nicht eindeutig. Wenn mehrere Personen die Annotation vornehmen, kann es auch sein, dass sie unterschiedlich viel Erfahrung haben.

In [22, S. 73 f.] werden verschiedene Methoden besprochen trotzdem ein gutes Modell zu lernen. Eine Möglichkeit ist, dass der Computer erkennt, wenn eine Annotation nicht zu dem Rest passt und den Experten erneut zu fragen, um mögliche Fehler zu korrigieren.

Hier stellt sich die Frage, wann ein Beispiel erneut abgefragt werden soll. In [25] werden hierzu verschiedene Möglichkeiten getestet.

Wenn mehrere Personen die Daten annotieren und angeben, wie sicher sie sich sind, kann der Computer versuchen Beispiele, die nicht sicher annotiert wurden, noch einmal von einer anderen Person annotieren zu lassen. In [27] wird dieses Szenario, das multiple expert active learning (MEAL) genannt wird untersucht. Die Idee ist, dass es nur wenig Experten mit viel Erfahrung gibt und diese beim Annotieren hohe Kosten verursachen, während es viele weniger erfahrene Personen gibt, die weniger Kosten verursachen. Um die Kosten zu minimieren sollten die meisten Beispiele von den weniger erfahrenen Personen annotiert werden, aber wenn diese für ein Beispiel nur eine unsichere Annotation liefern, soll ein Experte gefragt werden.

3.4 Wiederverwenden der Daten

Es kann sein, dass man später die Verfahren zum Klassifizieren ändern will, um bessere Ergebnisse zu erzielen. Man könnte z. B. den Kernel der SVM ändern oder die SVM durch ein anderes Verfahren ersetzen. Da das Annotieren der Daten teuer ist, würde man die Daten gerne wiederverwenden. Durch das aktive Lernen entspricht die Verteilung der Daten aber nicht mehr der tatsächlichen Verteilung. Das kann zu Problemen führen, wenn das neue Verfahren die Verteilung benutzt oder auf Häufigkeiten zurückgreift.

In [22, S. 76 f.] werden verschiedene Situationen besprochen, in denen trotzdem die Daten wiederverwendet werden können. In dieser Arbeit wurde die Wiederverwendbarkeit der ausgewählten Daten aber nicht getestet.

Kapitel 4

Implementierung in RapidMiner

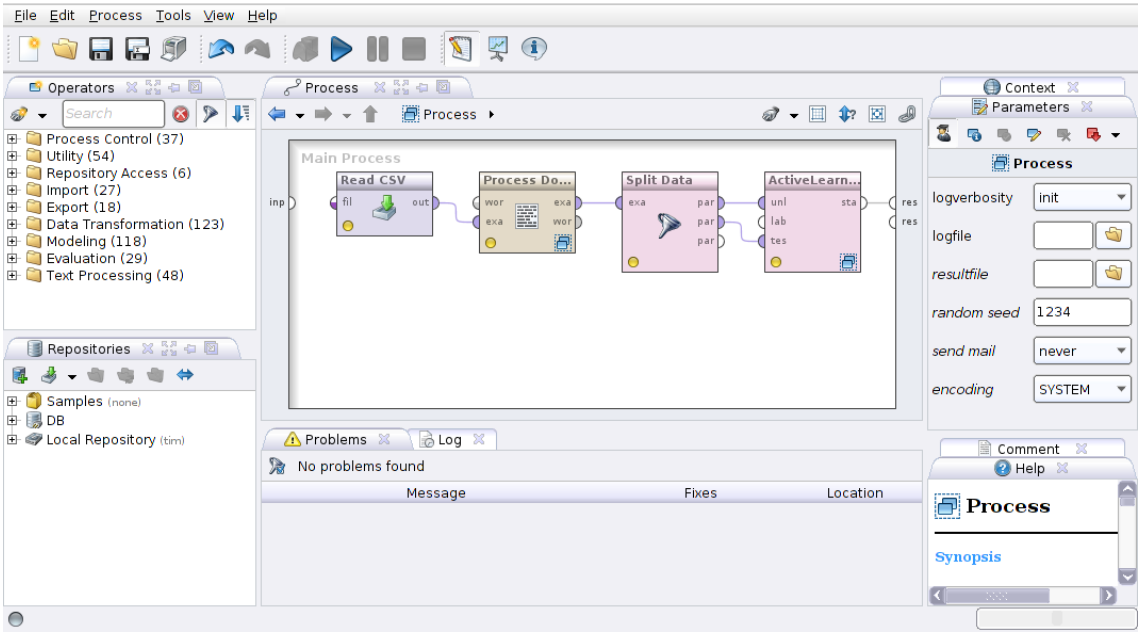

Abbildung 4.1: RapidMiner 5.3 mit einem Experiment zu aktivem Lernen als Prozess

4.1 RapidMiner

Zur Implementierung wird RapidMiner 5.3 benutzt. Hier sind schon verschiedene Mechanismen zum Verarbeiten und Lernen der Daten vorhanden. In RapidMiner werden Prozesse grafisch modelliert. In Abbildung 4.1 sieht man RapidMiner mit einem Beispiel-Prozess. Ein Prozess besteht aus Operatoren, die die Daten an ihren Eingängen verarbeiten und die Ergebnisse an ihre Ausgänge geben. Manche Operatoren kommen auch ohne Eingänge aus und beziehen die Daten aus externen Quellen wie z. B. Dateien. Operatoren können über Parameter genauer konfiguriert werden, die rechts neben dem Prozess dargestellt wer-

den. Es gibt auch Operatoren, die eigene Unterprozesse haben, die bei der Ausführung der Operatoren auch ausgeführt werden können. Ein Beispiel hierfür sind Schleifen.

RapidMiner lässt sich über Plugins erweitern [21], die neue Operatoren zur Verfügung stellen. Für diese Arbeit wurde auch ein Plugin mit weiteren Operatoren zu aktivem Lernen geschrieben. Das Plugin wurde dabei genau wie RapidMiner in Java geschrieben. Ein weiteres schon existierendes Plugin, das für die Experimente in Kapitel 5 benutzt wurde ist die Text Processing Extension [20]. Hiermit lassen sich Texte in Wortvektoren umwandeln, sodass sie zum Lernen verwendet werden können. Das Lernen selbst wird über den schon in RapidMiner vorhandenen Operator für die libSVM [9] gemacht.

4.1.1 ExampleSets

ExampleSet (10 examples, 2 special attributes, 510 regular attributes)

Row No.	text	label	a	able	about	action	actually
1	shy of love	-1	0	0	0	0	0
2	write posit	-1	0.025	0.108	0	0	0
3	p under ov	-1	0.017	0	0.069	0	0.180
4	contends t	-1	0.019	0	0	0	0
5	not great (	-1	0.002	0	0	0	0
6	old classic	1	0.016	0.167	0.161	0	0
7	only only h	1	0.009	0	0.084	0	0
8	helpful hel	1	0.011	0	0	0	0
9	or possibili	1	0	0	0.176	0	0.153
10	tears veys	1	0.005	0	0.131	0	0

Abbildung 4.2: Darstellung eines *ExampleSet*s mit Text, Label und Worthäufigkeiten in RapidMiner

Zum Austausch von Daten zwischen den Operatoren wird im RapidMiner die Klasse *IOObject* benutzt, von der es mehrere Unterklassen gibt. Die Klasse *Model* wird z. B. benutzt, um das von der libSVM gelernte Modell zu repräsentieren. Die Text Processing Extension enthält außerdem weitere Klassen. Für die neu geschriebenen Operatoren zu aktivem Lernen ist aber nur die Klasse *ExampleSet* relevant.

Ein *ExampleSet* enthält eine Menge von Objekten der Klasse *Example*. Außerdem gehören zu einem *ExampleSet* *Attribute*, die beschreiben, welche Daten für jedes Beispiel gespeichert werden. Die Attribute werden in reguläre und spezielle eingeteilt. Reguläre Attribute enthalten die Daten, die zum Lernen benutzt werden können. Spezielle Attribute enthalten z. B. das Label oder die Vorhersage des Modells, sowie die Konfidenzen.

In Abbildung 4.2 ist die grafische Darstellung eines *ExampleSet*s in RapidMiner zu sehen. Hier gibt es die zwei speziellen Attribute text und label, sowie 510 reguläre Attribute. Die Attribute text und label wurden dabei aus einer Datei gelesen und die regulären Attribute wurden von dem Tokenize-Operator der Text Processing Extension aus dem

text-Attribut berechnet. Abbildung 4.3 zeigt ein ähnliches *ExampleSet* mit zusätzlichen Attributen, die durch das Anwenden eines vorher gelernten Modells erzeugt wurden.

4.2 Operatoren

Es wurden drei neue Operatoren implementiert. Der wichtigste ist *ActiveLearningSample*. Mit ihm werden neue Beispiele ausgewählt, die annotiert und dann zum Lernen benutzt werden. Der Operator *ActiveLearning* führt die Unterprozesse in einer Schleife aus, um das Modell nach und nach zu verbessern. Mit dem Operator *AverageLoop* kann man das aktive Lernen mehrfach ausführen, um gemittelte Ergebnisse zu erhalten.

In Anhang A (Seite 38) wird die Benutzung der Operatoren genauer beschreiben. Hier wird auf die Implementierung eingegangen.

4.2.1 Sampling mit Konfidenz

ActiveLearningSample	
Eingang	Ausgang
example set input	example set output
	original
	unused

Tabelle 4.1: Ein- und Ausgänge des ActiveLearningSample-Operators

Der Operator *ActiveLearningSample* wird benutzt, um neue Beispiele auszuwählen. In Tabelle 4.1 sind alle Eingänge und Ausgänge aufgeführt. Der Operator erbt von der Klasse *AbstractSamplingOperator*, die bereits einen Eingang für die zur Verfügung stehenden Daten und zwei Ausgänge enthält. Außerdem hat er schon eine *doWork*-Methode implementiert, die die Daten am Eingang entgegen nimmt und an die neue *apply*-Methode zum Auswählen weitergibt, sowie das Ergebnis am Ausgang *example set output* ausgibt. Außerdem werden die ursprünglichen Daten an Ausgang *original* gegeben.

Zusätzlich zu den schon vorhandenen Ausgängen hat *ActiveLearningSample* noch den dritten Ausgang *unused*, der die nicht ausgewählten Beispiele enthält. Er wird benutzt, um die nicht ausgewählten Beispiele in späteren Runden benutzen zu können.

Mit dem Parameter *sample size* kann die maximale Anzahl neuer Beispiele pro Runde angegeben werden. Wenn der Parameter *use_confidence* aktiviert ist und die Beispiele am Eingang des Operators mit Konfidenzen versehen sind, dann werden diese zur Auswahl benutzt. Ansonsten werden die Beispiele zufällig ausgewählt. Abbildung 4.3 zeigt ein *ExampleSet* mit Konfidenzen. Diese sind in den Attributen "confidence(-1)" und "confidence(1)" enthalten.

ExampleSet (1000 examples, 5 special attributes, 510 regular attributes)

Row No.	text	label	confidence(-1)	confidence(1)	prediction...	a	able	about	action
1	great grea	1	0.426	0.574	1	0.015	0	0	0
2	crappy it re	-1	0.608	0.392	-1	0.010	0	0	0
3	like never h	1	0.434	0.566	1	0.012	0.227	0.073	0
4	found stale	-1	0.546	0.454	-1	0.007	0	0.119	0
5	itself indee	-1	0.550	0.450	-1	0.020	0	0	0
6	an answer	-1	0.584	0.416	-1	0	0	0	0
7	board she	1	0.546	0.454	-1	0.022	0.079	0.025	0.090
8	was possib	-1	0.440	0.560	1	0	0	0	0
9	silver eagle	1	0.508	0.492	-1	0.021	0	0	0
10	god s word	-1	0.504	0.496	-1	0.003	0	0	0

Abbildung 4.3: Darstellung eines ExampleSets mit Text, Label, Konfidenzen, Vorhersage und Worthäufigkeiten in RapidMiner

Um die Konfidenzen in dem Operator zu benutzen, müssen zuerst diese Attribute gefunden werden. Der Name eines Konfidenz-Attributes beginnt immer mit "confidence_". Wenn ein solches Attribut gefunden wurde, kann es für die Auswahl der Beispiele benutzt werden.

Da der Operator Zufall benutzt, hat er außerdem Parameter zum festlegen eines Seeds.

4.2.2 ActiveLearning

ActiveLearning

Eingang	Training		Apply		Performance		Ausgang
	Ausgang	Eingang	Ausgang	Eingang	Ausgang	Eingang	
unlabeled data	labeled data	model	model	labeled data	labeled data	performance	statistics
labeled data	unlabeled data	labeled data	unlabeled data				
test data		unlabeled data	through				
		parameters					
		through					

Tabelle 4.2: Ein- und Ausgänge des ActiveLearning-Operators und der Unterprozesse

Auch wenn RapidMiner bereits verschiedene *Loop*-Operatoren hat, mit denen die Schleife für das aktive Lernen implementiert werden kann, würde ein solcher RapidMiner-Prozess sehr komplex werden. Deshalb gibt es den Operator *ActiveLearning*. Er führt die Schritte des aktiven Lernens, die in Unterprozessen definiert sind, in einer Schleife aus und erstellt dabei eine Tabelle mit Statistiken für jeden Schleifendurchlauf. Zusätzlich ist es möglich die Schleife mit vorgegebenen Abbruchkriterien vorzeitig zu beenden.

Der Operator hat außerdem Parameter, um ein Makro für die aktuelle Iteration zu definieren. Die Parameter entsprechen denen des *Loop*-Operators von RapidMiner.

Tabelle 4.2 zeigt alle Ein- und Ausgänge des Operators und der Unterprozesse. Der Operator benötigt immer ein *ExampleSet* an Eingang *unlabeled data*, aus dem Beispiele ausgewählt werden können. Zusätzlich können bereits annotierte Daten an *labeled data* und Daten zum Testen an *test data* gegeben werden. Es gibt drei Unterprozesse. Abbildung A.3 (Seite 39) zeigt, wie die Unterprozesse für ein Experiment ohne das Annotieren durch einen Menschen aussieht.

In Unterprozess *Training* werden neue Beispiele ausgewählt. Wenn die Daten noch kein Label haben, muss sie hier auch noch annotiert werden. Außerdem wird ein neues Modell mit allen annotierten Daten gelernt.

Der Unterprozess *Apply* wird benutzt, um das gelernte Modell auf neue Daten anzuwenden. Hier werden die Konfidenzen ermittelt, die in der nächsten Runde zur Auswahl der Beispiele benutzt werden. Wenn es Testdaten gibt, dann wird das Modell auch auf diese angewendet und der Unterprozess *Performance* wird benutzt, um die Vorhersage des Modells zu bewerten.

ExampleSet (10 examples, 0 special attributes, 6 regular attributes)

Row No.	labeled	labeled -1	labeled 1	unlabeled	accuracy	minconfdiff
1	100	47	53	900	0.610	0.000
2	200	104	96	800	0.700	0.000
3	300	159	141	700	0.741	0.000
4	400	208	192	600	0.764	0.000
5	500	257	243	500	0.764	0.000
6	600	303	297	400	0.774	0.003
7	700	345	355	300	0.775	0.007
8	800	399	401	200	0.785	0.074
9	900	452	448	100	0.780	0.145
10	1000	500	500	0	0.787	∞

Abbildung 4.4: Tabelle mit Statistiken, die durch den ActiveLearning-Operator erzeugt wird

Die Schleife ist in der *doWork*-Methode des Operators implementiert. Hier werden die Unterprozesse aufgerufen. Zum Erstellen der Statistik wird die Hilfsklasse *ActiveLearning.StatisticsBuilder* benutzt. Sie enthält die Methode *add* mit der in jedem Schleifendurchlauf eine neue Zeile zu der Tabelle hinzugefügt wird. Beim ersten Aufruf von *add* wird außerdem bestimmt, welche Attribute die Tabelle haben soll. In Abbildung 4.4 ist zu sehen, wie das Ergebnis aussehen kann. Die Statistik enthält Attribute für die Anzahl der Beispiele mit Label, mit einem bestimmten Label und ohne Label, sowie den Wert *minconfdiff*, der für das Abbruchkriterium benutzt wird. Wenn es Testdaten gibt, wird außerdem das Ergebnis des *Performance*-Unterprozesses angegeben.

Über die Parameter des Operators lässt sich bestimmen, wann die Schleife abgebrochen werden soll. Wenn keine Beispiele mehr zur Auswahl stehen, wird immer abgebrochen. Mit dem Parameter *max iterations* lässt angeben, wie viele Runden maximal durchgeführt werden sollen. Über den Parameter *min confidence* kann man die Konfidenz für das Abbruch-

kriterium benutzen. Es wird dann abgebrochen, wenn der kleinste Abstand der Konfidenz der Beispiele von 0.5 über diesem Wert liegt.

4.2.3 AverageLoop

AverageLoop			
Eingang	inner		Ausgang
	Ausgang	Eingang	
input	input	example set	example set

Tabelle 4.3: Ein- und Ausgänge des AverageLoop-Operators

Da die Auswahl der Beispiele mindestens im ersten Schleifendurchlauf immer zufällig erfolgt, ist auch die Statistik des *ActiveLearning*-Operators vom Zufall abhängig. Um in den Experimenten trotzdem Ergebnisse zu erhalten, die weniger zufällig sind, gibt es den Operator *AverageLoop*. In Tabelle 4.3 sind alle Ein- und Ausgänge aufgeführt. Er enthält einen Unterprozess, in dem der *ActiveLearning*-Operator eingefügt werden kann. Der wird dann mehrfach mit der Eingabe des *AverageLoop*-Operators ausgeführt und es wird der Durchschnitt der Ergebnisse gebildet. Genau wie der *ActiveLearning*-Operator hat auch dieser Operator Parameter zum Setzen eines Makros für die aktuelle Iteration.

Die Funktionalität des Operators ist komplett in der *doWork*-Methode enthalten. Hier wird der Unterprozess so oft aufgerufen, wie in Parameter *num iterations* angegeben. Aus allen Ergebnissen wird dann der Durchschnitt gebildet und an den Ausgang des Operators gegeben.

Kapitel 5

Experimente

5.1 Benutzte Daten

Um zu überprüfen, ob aktives Lernen mit dem implementierten Plugin besser als zufälliges Wählen der Beispiele ist, wurden mehrere Experimente durchgeführt. Dazu wurden vier Datensätze mit Amazon Reviews [5] benutzt. Dabei gibt es die vier verschiedenen Datensätze books, kitchen, electronics und dvd. In jedem sind jeweils 1000 positive und 1000 negative Beispiele genommen.

Jedes Beispiel enthält den Text eines Reviews und die Bewertung als Annotation, wobei die Annotation positiv oder negativ ist.

5.2 Versuchsaufbau

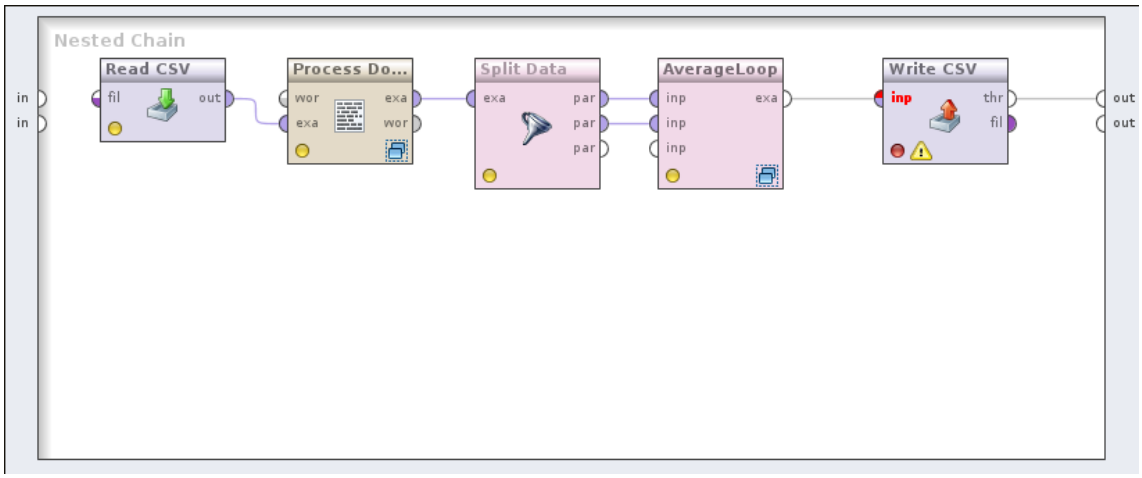

Abbildung 5.1: RapidMiner-Prozess eines Experiments

Alle Experimente wurden mit einem RapidMiner-Prozess, wie in Abbildung 5.1, durchgeführt. Zunächst werden die Daten über den Operator *Read CSV* eingelesen und man

erhält ein ExampleSet mit Text und Label. Danach werden die Texte mithilfe der Text Processing Extension [20] in Wortvektoren umgewandelt, damit das in der Schleife nicht in jedem Durchlauf wiederholt werden muss. Dazu wird der Operator *Prozess Documents from Data* benutzt, der für jedes Beispiel im *ExampleSet* seinen Unterprozess und den Text dabei in ein *Document* umwandelt. In dem Unterprozess wird das *Document* dann mit dem Operator *Tokenize* in die Wörter aufgespaltet. Nach Ausführen des Unterprozesses für alle Texte wird ein *ExampleSet* erzeugt, das neben dem Label und dem Text auf Attribute für alle möglichen Wörter mit den relativen Häufigkeiten hat.

Danach werden die Beispiele mit dem *Split Data*-Operator gleichmäßig auf Trainings- und Testdaten aufgeteilt. Beide Mengen werden dann an den Operator *AverageLoop* weitergegeben, der wiederum den Operator *ActiveLearning* enthält und ihn 10 mal aufruft, um gemittelte Ergebnisse zu erzeugen. Die Aufteilung in Trainings- und Testdaten ist also für alle Iteration des *AverageLoop*-Operators gleich. Der Seed dieses *Split Data*-Operator ist 1992.

Die Unterpozesse des *ActiveLearning* sind wie im Anhang (Seite 38) beschrieben aufgebaut. Zum Lernen wird der Operator der libSVM mit dem RBF-Kernel verwendet.

Der Parameter *use_ confidence* des Operators *ActiveLearningSample* wurde für jedes Experiment einmal aktiviert und einmal deaktiviert, um die Ergebnisse zu vergleichen. Die Zahl der Beispiele pro Runde unterscheidet sich zwischen den Experimenten.

Die Operatoren, die zufällige Ergebnisse erzeugen, haben einen Seed bekommen, der nur von der aktuellen Iteration der Operatoren *ActiveLearning* und *AverageLoop* abhängt. Bei allen Experimenten betrifft das den *ActiveLearningSample*-Operator. Dazu schreibt der *AverageLoop*-Operator die aktuelle Iteration startend bei 1 in das Makro *averageiteration* und der *ActiveLearning*-Operator schreibt die aktuelle Iteration startend bei 1 in das Makro *aliteration*. Mit dem Operator *Generate Macro* wird der benutzte Seed dann als $\%\{averageiteration\} * 1000 + \%\{aliteration\}$ berechnet. Da der Seed nicht davon abhängt, ob aktives Lernen benutzt wird, sind die Ergebnisse für die erste Menge an zufälligen Beispielen immer gleich.

Um das Ergebnis grafisch darzustellen, wurde jeweils ein Diagramm generiert, das für beide Varianten des Experiments einen Graph enthält, der die Accuracy in Abhängigkeit der Anzahl an Beispielen, die zum Lernen benutzt wurden, zeigt. Die Accuracy ist dabei der Anteil an Beispielen der Testmenge, die von der SVM richtig klassifiziert wurde.

5.3 Ergebnisse

5.3.1 100 Beispiele pro Durchlauf

Im ersten Versuch wurden pro Schleifendurchlauf 100 neue Beispiele ausgewählt. Es gibt also 10 Schleifendurchläufe, wobei bei dem ersten Durchlauf die Beispiele immer zufällig gewählt werden.

Bei allen Datensätzen sind sich aktives Lernen und zufälliges Auswählen der Beispiele sehr ähnlich. Für den Datensatz books (Abbildung 5.3, S. 29) sind die Ergebnisse bis zu 300 ausgewählten Beispielen gleich und ab 400 Beispielen ist aktives Lernen besser.

Bei den Datensätzen electronics (Abbildung 5.4, S. 29) und kitchen (Abbildung 5.5, S. 30) ist die Accuracy bei aktivem Lernen für 200 Beispiele schlechter und ab 400 Beispielen besser. Bei dem Datensatz dvd (Abbildung 5.6, S. 30) gibt es nur sehr geringe Unterschiede.

Da der Seed der Operator nicht davon abhängt, ob aktives Lernen benutzt wurde, sind die Ergebnisse für 100 Beispiele immer gleich. Man kann an den Graphen auch sehen, dass am Ende, wenn alle Beispiele ausgewählt wurden, wieder die gleiche Accuracy für aktives Lernen und zufälliges Auswählen erreicht wird.

5.3.2 10 Beispiele pro Durchlauf

Der nächste Versuch unterscheidet sich vom ersten nur darin, dass jetzt pro Runde 10 Beispiele ausgewählt werden. Damit gibt es Werte für 100 Schleifendurchläufe.

Hier ist die Accuracy für alle Datensätze (Seite 31f) bei aktivem Lernen genau so gut oder besser als zufälliges Auswählen der Beispiele. Nur bei dem Datensatz dvd (Abbildung 5.10, S. 32) ist für 500 bis 800 Beispiele die Accuracy schlechter. Aktives Lernen verbessert die Accuracy für den Datensatz electronics (Abbildung 5.8, S. 31) am meisten.

5.3.3 1 Beispiel pro Durchlauf

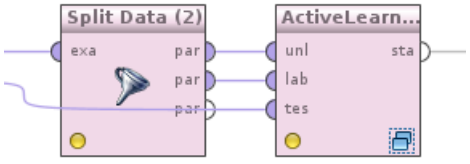

Abbildung 5.2: Vor dem ActiveLearning-Operator werden schon 50 Beispiele zufällig ausgewählt

Um die Ergebnisse weiter zu verbessern, wird jetzt nur ein Beispiel pro Runde ausgewählt. Damit zu Beginn schon genug Beispiele vorhanden sind, um ein Modell zu lernen, werden zuerst 50 Beispiele zufällig ausgewählt. Da das Lernen jetzt sehr aufwendig ist, wird die Anzahl der Runden für das aktive Lernen auf 350 beschränkt. Insgesamt werden also nur 400 Beispiele zum Lernen benutzt.

Um in der ersten Runde mehr Beispiele auszuwählen, als in den anderen Runden, muss der RapidMiner-Prozess weiter angepasst werden. Vor dem *ActiveLearning*-Operator muss mit einem weiteren *Split Data*-Operator eine zufällige Menge von 50 Beispielen ausgewählt werden, wie an Abbildung 5.2 zu sehen. Auch der Seed des *Split Data*-Operator hängt von der Iteration des *AverageLoop* ab. Er wird als $\%\{averageiteration\} * 1000 + 1$ festgelegt.

An den Graphen (Seite 33f) kann man sehen, dass aktives Lernen hier immer so gut oder besser als zufälliges Auswählen der Beispiele ist. Die Graphen ähneln dabei stark den Graphen für 10 Beispiele pro Runde, allerdings kann der Bereich für mehr als 400 Beispiele nicht verglichen werden.

5.3.4 Vergleich

Zum Vergleichen der Experimente ist wichtig, dass im letzten Experiment nur 400 Beispiele und nicht alle 1000 Beispiele ausgewählt wurden. Beim Vergleich mit den anderen Experimenten sollten also auch nur die ersten 400 Beispiele betrachtet werden. Außerdem hat aktives Lernen in den Experimenten weniger Auswahl an Beispielen, je mehr Beispiele schon ausgewählt wurden. Wenn alle 1000 Beispiele ausgewählt wurden, sind aktives Lernen und zufälliges Auswählen wieder gleich. Die Werte für viele Beispiele sind deshalb für aktives Lernen weniger relevant.

Um die Ergebnisse der einzelnen Experimente besser vergleichen zu können, wird für jedes Experiment berechnet, wie stark sich die Accuracy der Vorhersage bei aktivem Lernen gegenüber zufälligem Auswählen durchschnittlich für 50 bis 400 Beispiele erhöht. In der folgenden Tabelle sieht man für die vier Datensätze und die drei Experimente diese Erhöhung. Ein positiver Wert bedeutet dabei, dass aktives Lernen zu einer Verbesserung geführt hat, während ein negativer Wert bedeutet, dass aktives Lernen schlechter als zufälliges Auswählen ist.

Beispiele/Runde	books	kitchen	electronics	dvd
100	0.25629%	-0.320749%	-0.234719%	0.203186%
10	1.7704%	2.48256%	3.99613%	1.88629%
1	1.26868%	0.750762%	1.73997%	0.0811755%

Tabelle 5.1: Durchschnittliche Erhöhung der Accuracy für 50 bis 400 Beispiele

Bei den Datensätzen kitchen und electronics ist aktives Lernen schlechter als zufälliges Auswählen der Beispiele. Am besten sind die Ergebnisse für 10 Beispiele pro Runde. Allerdings wurden bei 10 Beispielen pro Runde nur die 10 Beispiele zufällig ausgewählt, während bei einem Beispiel pro Runde 50 Beispiele zufällig ausgewählt wurden. Da nur von 10 Durchläufen der Durchschnitt genommen wurde, können die Unterschiede auch noch stark vom Zufall abhängen.

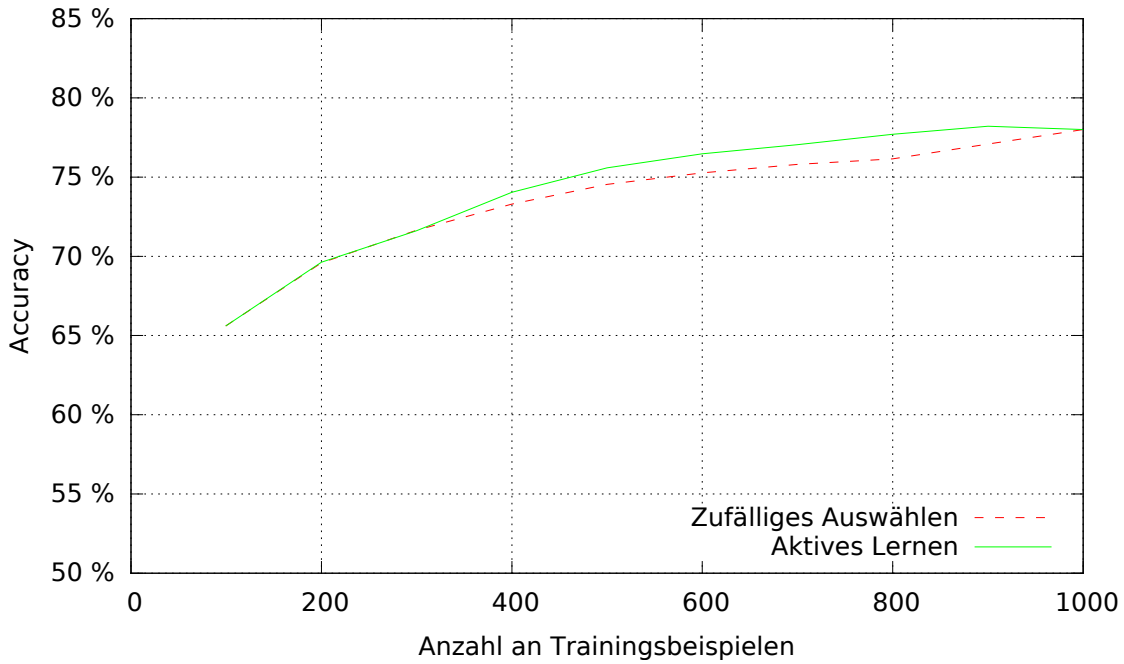

Abbildung 5.3: Ergebnisse für den Datensatz books, mit 100 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen ist hier leicht besser als zufälliges Auswählen

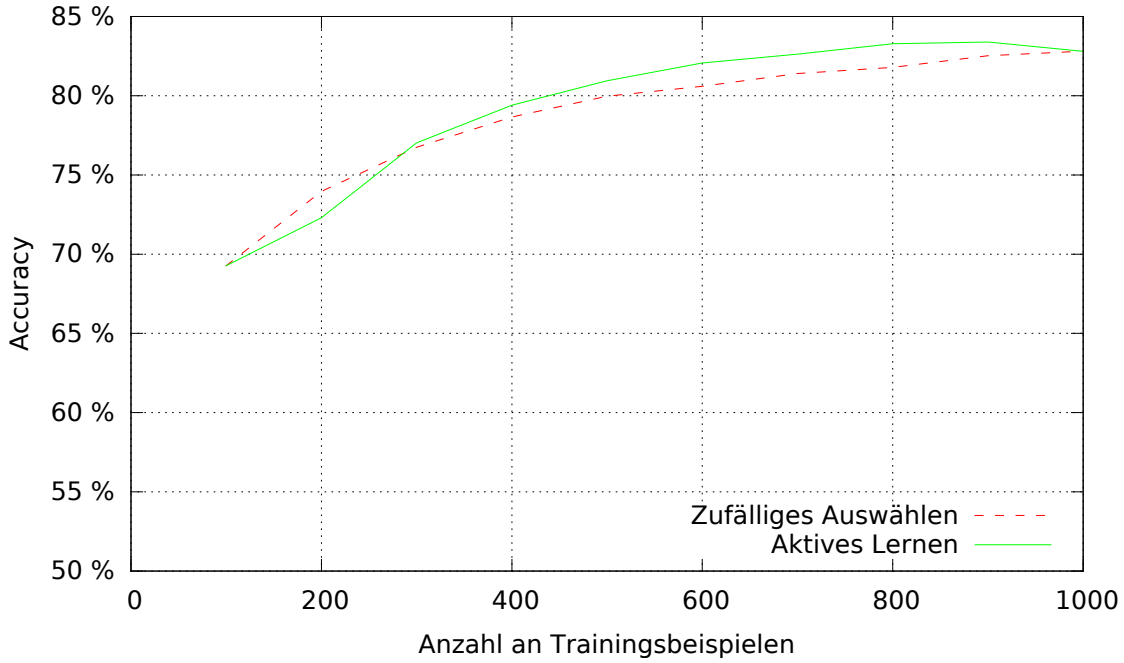

Abbildung 5.4: Ergebnisse für den Datensatz electronics, mit 100 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen ist hier zuerst schlechet und dann leicht besser als zufälliges Auswählen

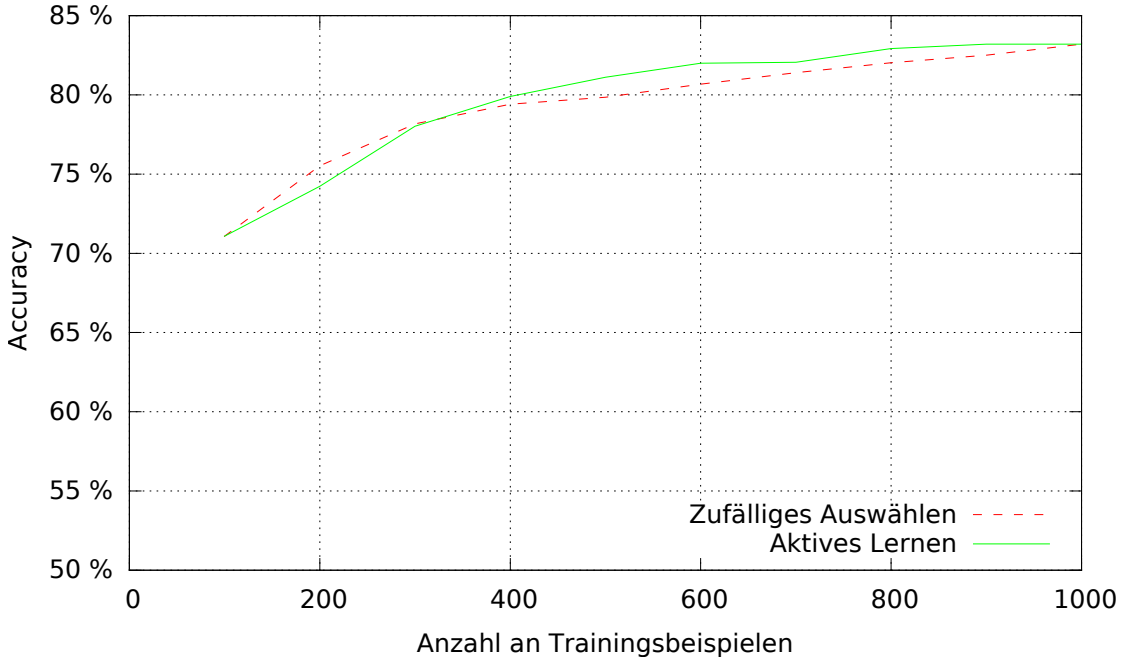

Abbildung 5.5: Ergebnisse für den Datensatz kitchen, mit 100 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen ist hier zuerst schlechter und später besser als zufälliges Auswählen

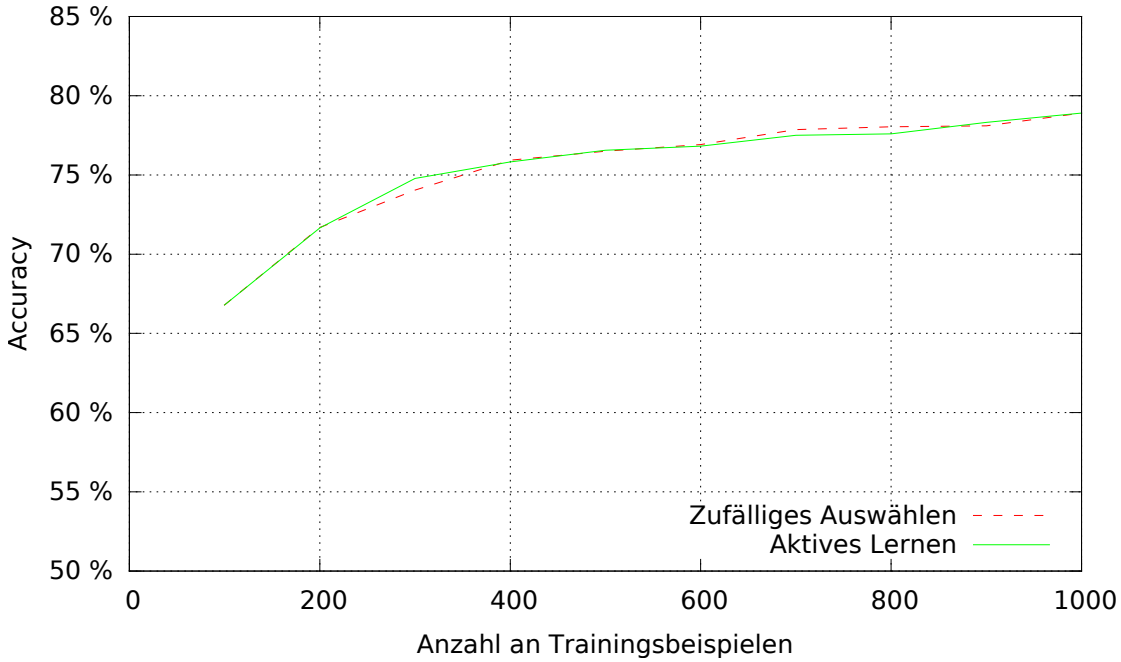

Abbildung 5.6: Ergebnisse für den Datensatz dvd, mit 100 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen unterscheidet sich hier kaum von zufälliges Auswählen

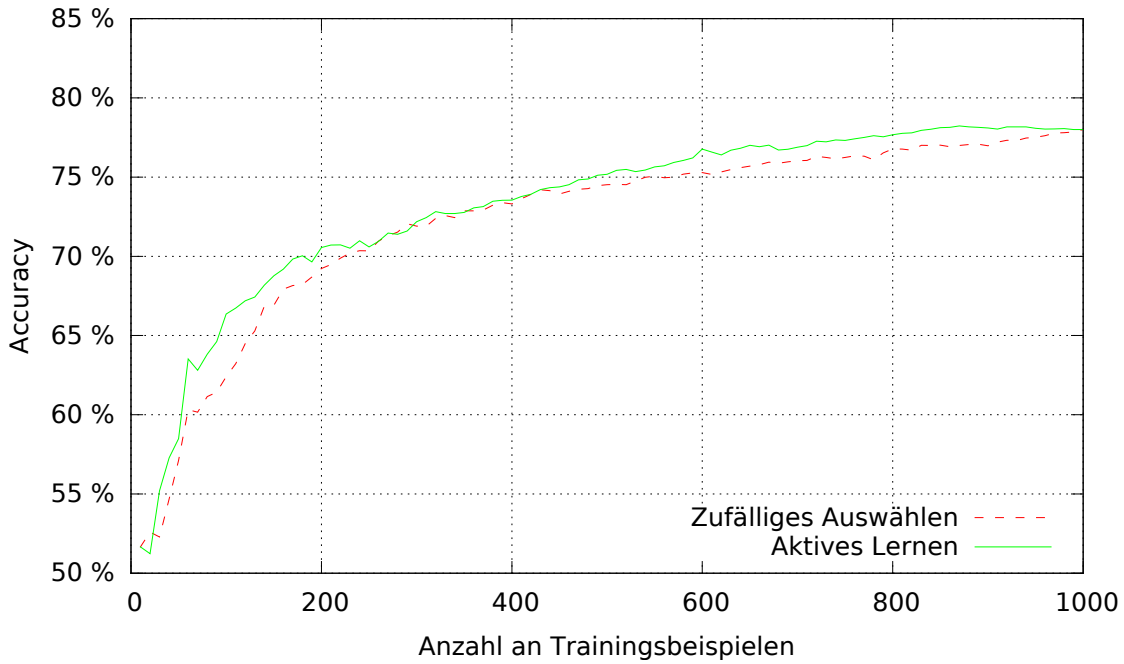

Abbildung 5.7: Ergebnisse für den Datensatz books, mit 10 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen ist hier teilweise leicht besser als zufälliges Auswählen

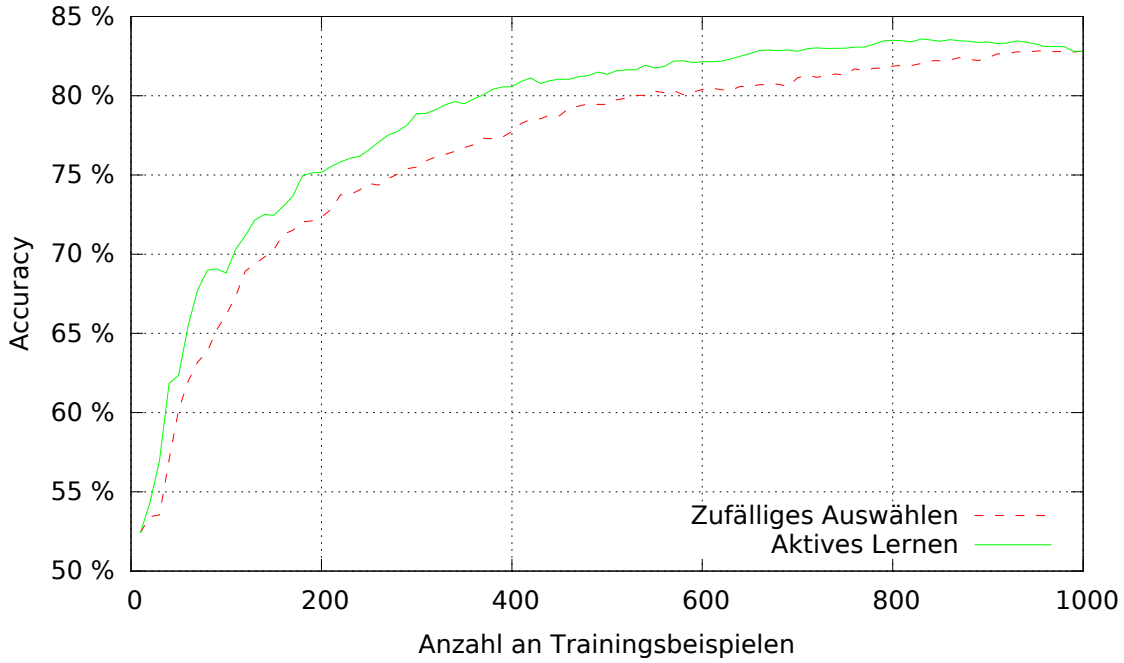

Abbildung 5.8: Ergebnisse für den Datensatz electronics, mit 10 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen ist hier besser als zufälliges Auswählen

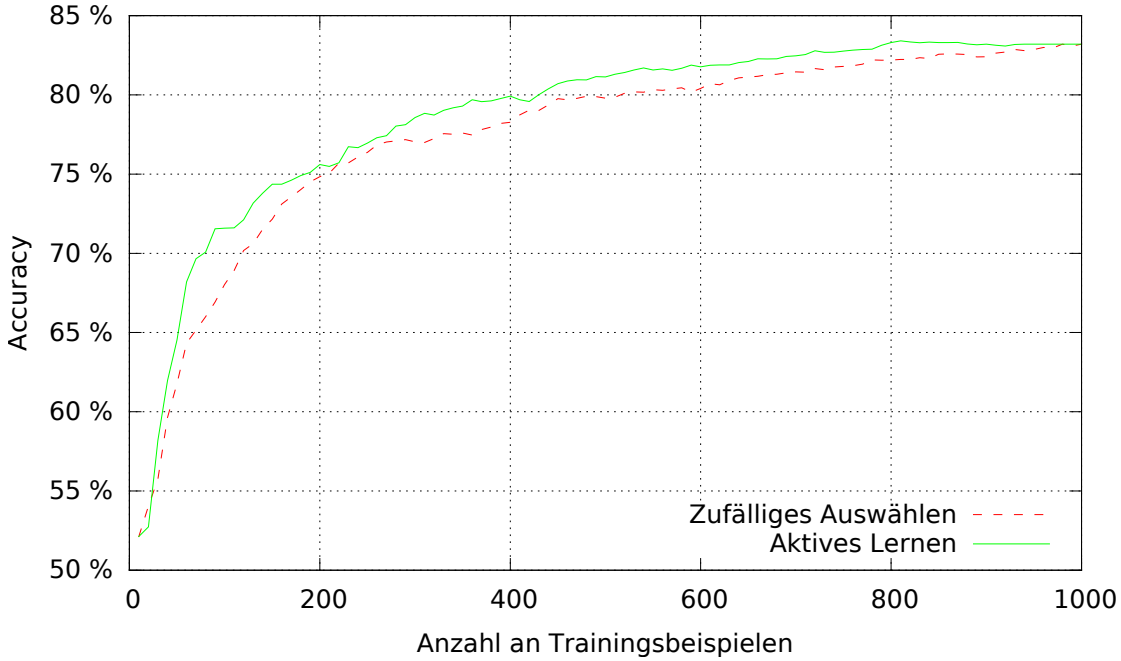

Abbildung 5.9: Ergebnisse für den Datensatz kitchen, mit 10 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen ist hier besser als zufälliges Auswählen

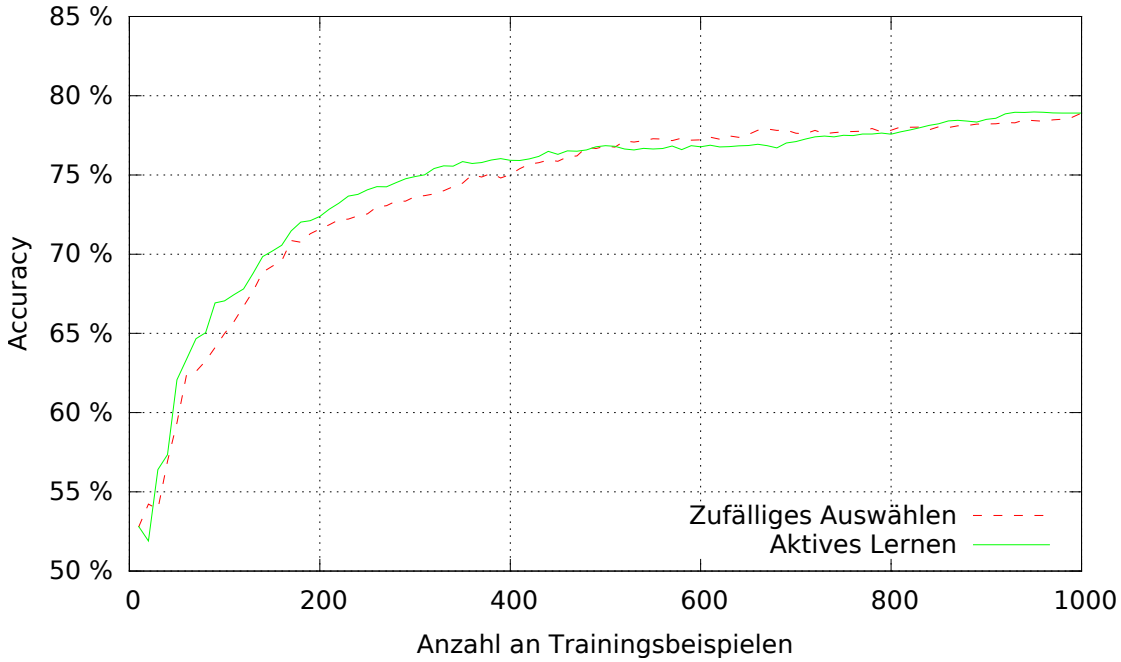

Abbildung 5.10: Ergebnisse für den Datensatz dvd, mit 10 neuen Beispielen pro Schleifendurchlauf
Aktives Lernen ist hier zuerst besser und dann schlechter als zufälliges Auswählen

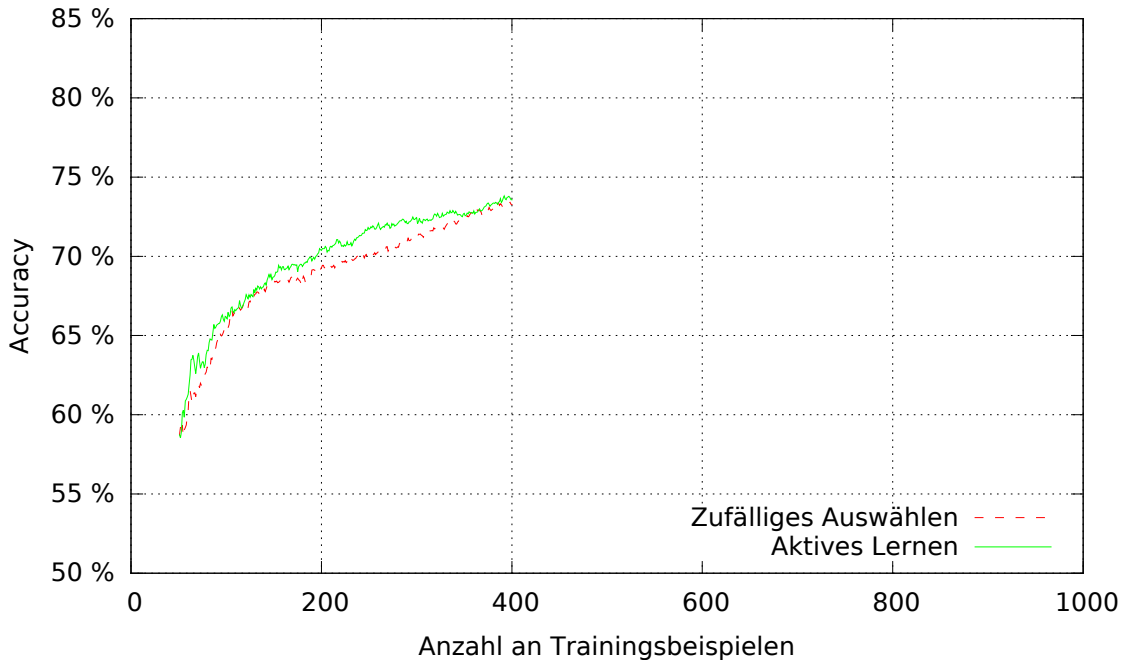

Abbildung 5.11: Ergebnisse für den Datensatz books, mit einem neuen Beispiel pro Schleifendurchlauf
Aktives Lernen ist hier teilweise besser als zufälliges Auswählen

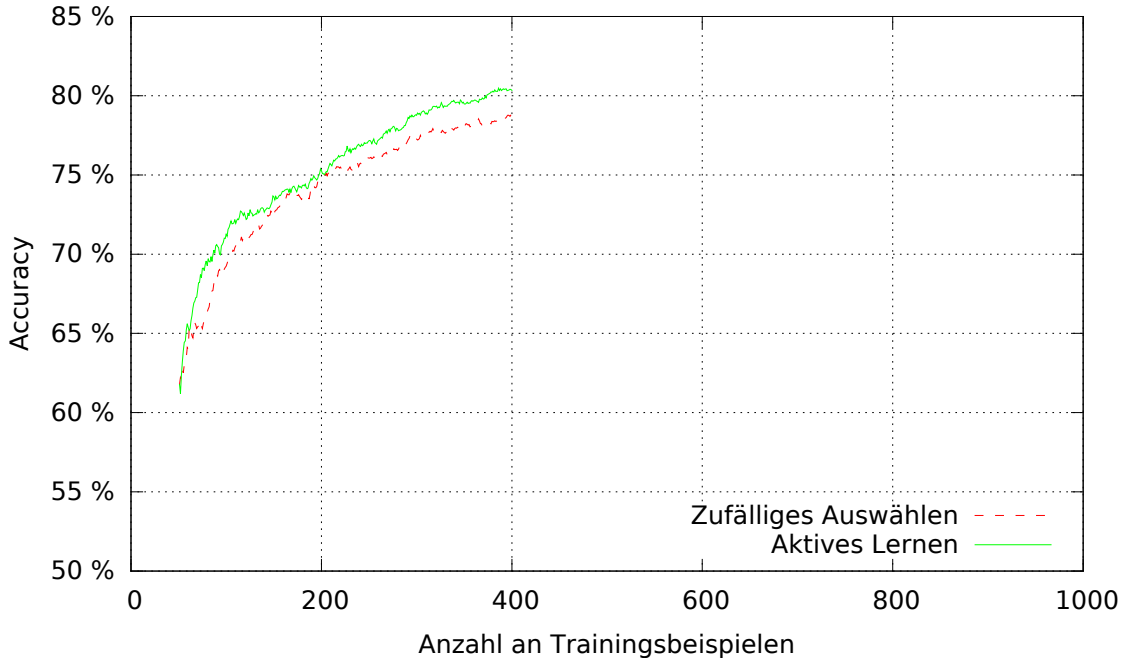

Abbildung 5.12: Ergebnisse für den Datensatz electronics, mit einem neuen Beispiel pro Schleifendurchlauf
Aktives Lernen ist hier besser als zufälliges Auswählen

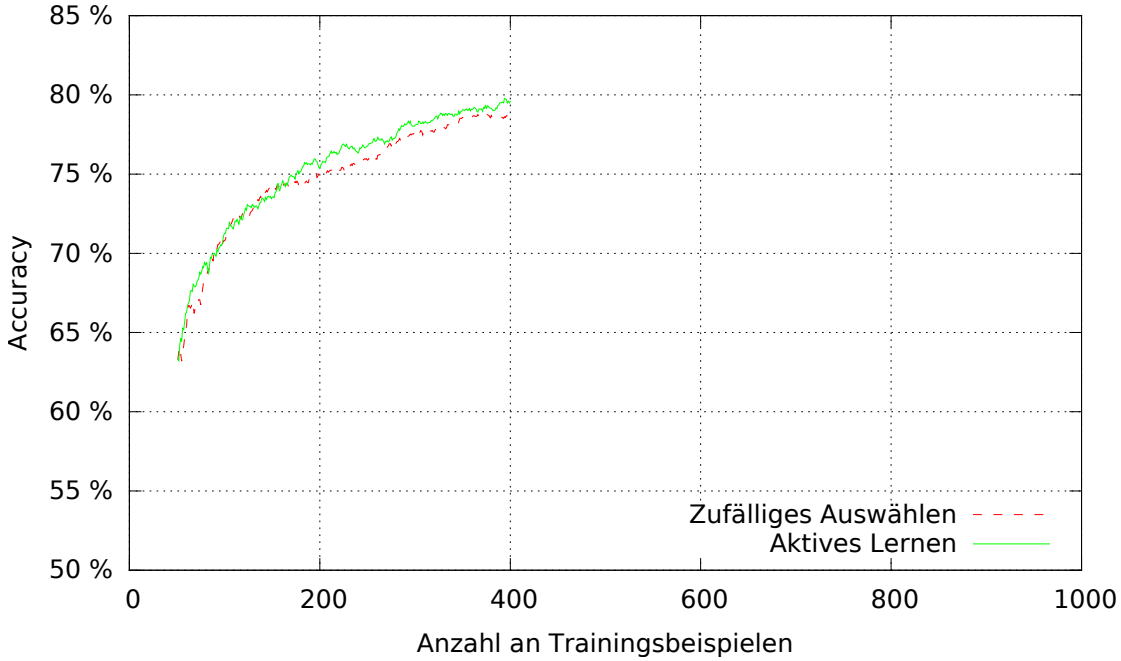

Abbildung 5.13: Ergebnisse für den Datensatz kitchen, mit einem neuen Beispiel pro Schleifendurchlauf
Aktives Lernen ist hier teilweise besser als zufälliges Auswählen

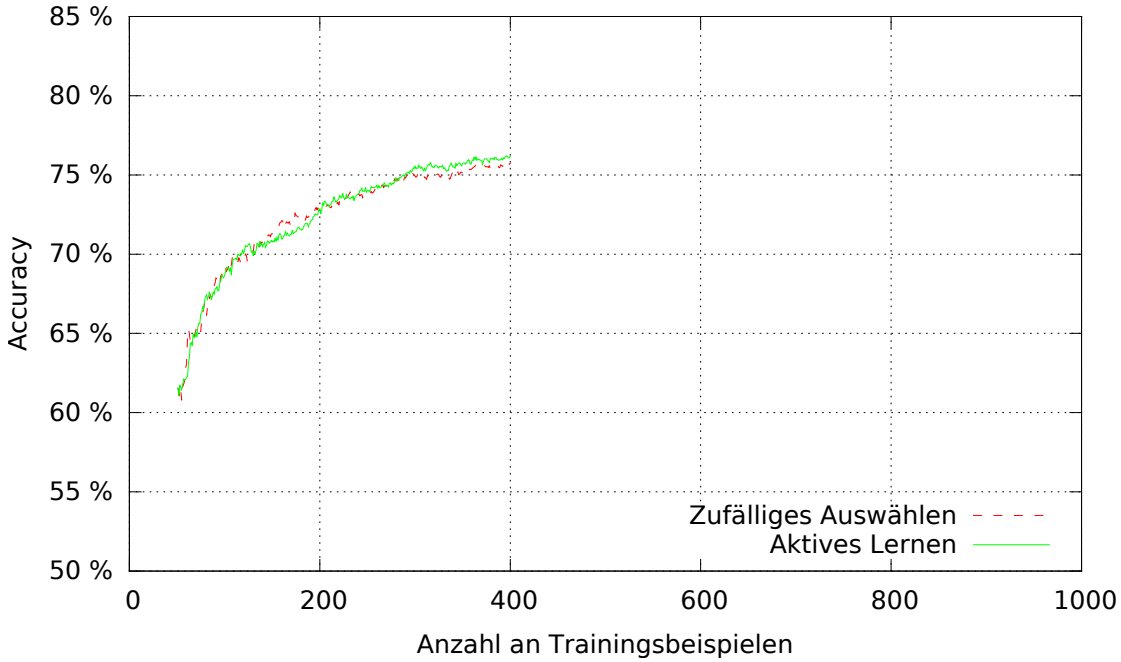

Abbildung 5.14: Ergebnisse für den Datensatz dvd, mit einem neuen Beispiel pro Schleifendurchlauf
Aktives Lernen unterscheidet sich hier kaum von zufälliges Auswählen

Kapitel 6

Folgerung

Mit dem erstellten Plugin für RapidMiner kann aktives Lernen benutzt werden, um die Zahl der Beispiele, die zum Lernen eines Modells notwendig sind, zu reduzieren. Die Experimente konnten für die benutzten Datensätze aber nur leichte oder keine Verbesserungen erzielen. Für 100 Beispiele pro Runde waren die Ergebnisse teilweise schlechter, weshalb man weniger Beispiele wählen sollte.

Eine Einschränkung ist hierbei die Laufzeit, wodurch mehrere Beispiele pro Runde ausgewählt werden müssen, um zu lange Wartezeiten zu vermeiden. In der Praxis muss man sich hier also entscheiden, wie viele Beispiele pro Runde am besten sind. Wenn man bei der Wahl der Beispiele die Ähnlichkeit der Beispiele berücksichtigen würde, könnte man hier eventuell weitere Verbesserungen erzielen.

Eine weitere mögliche Verbesserung wäre, die Parameter der SVM zu optimieren, was aber auch die Laufzeit verlängern würde. Außerdem bräuchte man eine Testmenge zum Optimieren, die nicht durch aktives Lernen gewählt wird. Es wäre auch möglich, die Dichte der Daten zu berücksichtigen, um nur relevante Beispiele auszuwählen.

Für die Experimente wurde immer der RBF-Kernel der libSVM benutzt. Man könnte hier auch andere Kernel benutzen, um die Ergebnisse zu verbessern. Wenn man eigene Kernel-Matrizen außerhalb der libSVM berechnet, könnte man die Laufzeit auch verbessern, indem man bereits berechnete Werte aus den vorherigen Runden wiederverwendet. In dem Fall, dass ein Benutzer die Annotation vornimmt könnten die Kernel-Werte auch schon berechnet werden, während der Benutzer die Beispiele annotiert.

Anhang A

Benutzung des Plugins

Nachdem das Plugin für RapidMiner installiert wurde, stehen zwei Operatoren zum aktiven Lernen zur Verfügung. Der Operator *ActiveLearningSample* wählt aus einer Menge von Beispielen neue Beispiele für die Trainingsdaten aus. Der Operator *ActiveLearning* führt in einer Schleife die Operatoren der Unterprozesse aus und erstellt eine Tabelle mit den Ergebnissen.

Außerdem gibt es den Operator *AverageLoop*, mit dem das aktive Lernen mehrfach ausgeführt und gemittelt werden kann.

A.1 ActiveLearningSample

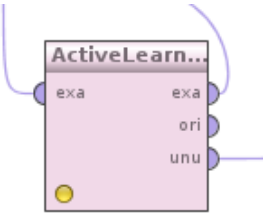

Abbildung A.1: Der ActiveLearningSample-Operator

Der Operator *ActiveLearningSample* hat einen Eingang, der die zur Auswahl stehenden Beispiele erhält. Es gibt drei Ausgänge. Der Ausgang *example set output* enthält die ausgewählten Beispiele. An Ausgang *original* werden die Daten des Eingangs unverändert weitergeleitet. Die nicht ausgewählten Daten werden an Ausgang *unused* gegeben.

Mit dem Parameter *sample size* des Operators lässt sich einstellen, wie viele Beispiele höchstens ausgewählt werden sollen. Wenn der Parameter *use confidence* gesetzt ist und die Beispiele am Eingang eine Konfidenz haben, dann wählt der Operator die Beispiele anhand der Konfidenz aus. Ansonsten werden die Beispiele zufällig ausgewählt.

A.2 ActiveLearning

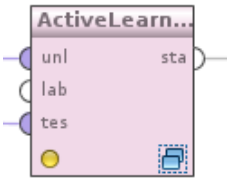

Abbildung A.2: Der ActiveLearning-Operator

Der *ActiveLearning*-Operator hat drei Eingänge und einen Ausgang. Der Eingang *unlabeled data* erhält die Daten, aus denen die Beispiele zum Lernen ausgewählt werden sollen.

Optional können an Eingang *labeled data* bereits annotierte Daten gegeben werden. Der Eingang *test data* erhält optional Beispiele, um nach jeder Runde zu überprüfen, wie gut das Modell diese Beispiele vorhersagt.

Der Ausgang *statistics* enthält eine Tabelle, in der für jede Iteration des Operators eine Zeile Daten über diese Iteration enthält. Die Spalten der Tabelle hängen von der Konfiguration der Unterprozesse und den Daten am Eingang ab. Es gibt immer die Spalten *labeled* für die Anzahl der bereits ausgewählten Beispiele, *unlabeled* für die Anzahl der noch zur Verfügung stehenden Daten und *minconfdiff* für den minimalen Abstand aller Konfidenzen der noch nicht ausgewählten Beispiele von 0.5. Außerdem gibt es für jeden möglichen Wert des Labels eine Spalte mit der Anzahl der Beispiele mit diesem Label in den ausgewählten Daten. Wenn am Eingang *test data* eine Testmenge angegeben wurde, dann wird außerdem eine Spalte für die Performance, die von einem Unterprozess berechnet wird, erzeugt.

Um zu steuern, wie viele Iterationen der Operator durchläuft, gibt es zwei Parameter, die beide optional sind. Der Parameter *max iterations* spezifiziert eine maximale Anzahl an Iterationen. Mit dem Parameter *min confidence* kann angegeben, dass abgebrochen wird, wenn der Wert *minconfdiff* größer als der angegebene Wert ist. Dieser Wert muss im Intervall von 0 bis 0.5 liegen.

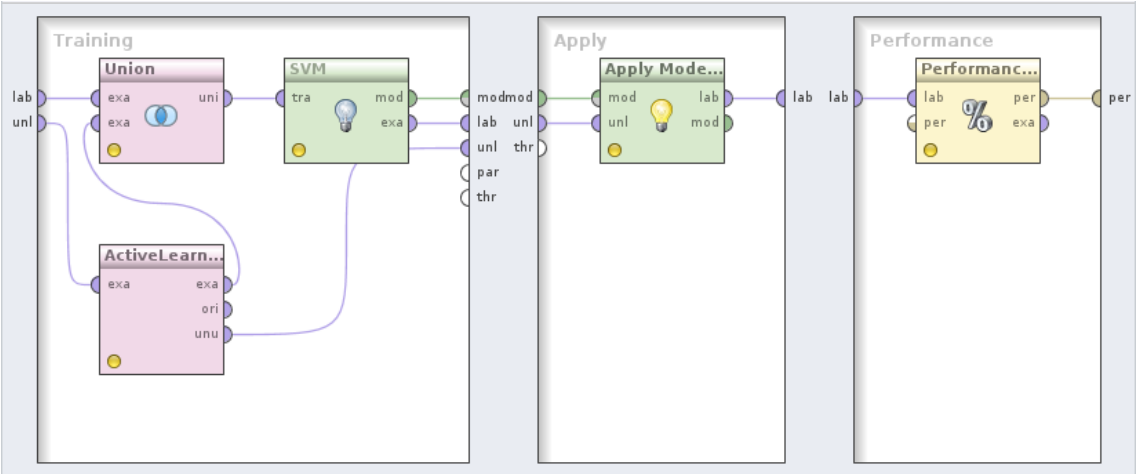

Abbildung A.3: Die Unterprozesse des ActiveLearning-Operators

Der Operator hat drei Unterprozesse. Im Unterprozess *Training* werden neue Beispiele ausgewählt und es wird ein Modell mit allen Daten trainiert. Der Unterprozess *Apply* spezifiziert, wie mit dem Modell neue Daten klassifiziert und mit einer Konfidenz versehen werden. Wenn der Eingang *test data* belegt ist, wird außerdem der Unterprozess *Performance* benutzt, um die Vorhersage zu bewerten.

A.2.1 Training

Der Unterprozess *Training* erhält die bereits ausgewählten Daten am Ausgang *labeled data* und die noch zur Verfügung stehenden Daten, die eventuell Konfidenzen enthalten, am Ausgang *unlabeled data*.

Zuerst werden die Daten von *unlabeled data* an den Operator *ActiveLearningSample* weitergegeben werden. Dieser gibt die nicht ausgewälten Daten am Ausgang *unused* aus und sie werden direkt an den Eingang *unlabeled data* des Unterprozesses weitergegeben. Die ausgewählten Daten können jetzt weiterverarbeitet werden, um ein Label zu erhalten und werden dann durch den *Union*-Operator mit den schon vorher ausgewählten Daten verknüpft. Das Ergebnis kann jetzt zunächst gespeichert werden und wird dann zum Trainieren eines Modells verwendet. Das Modell wird an Eingang *model* weitergegeben und die ausgewählten Daten werden an den Eingang *labeled data* gegeben.

Wenn beim Trainieren des Modells Parameter optimiert wurden, können diese an Eingang *parameters* gegeben werden und werden dann in der Tabelle zum Schluss aufgeführt. Über den Eingang *through* können beliebige zusätzliche Daten am den Unterprozess *Apply* weitergegeben werden.

A.2.2 Apply

Im Unterprozess *Apply* wird das Modell an Ausgang *model* auf die Daten an Ausgang *unlabeled data* angewendet. Dazu können die zusätzlichen Daten an Ausgang *through* benutzt werden. Das Ergebnis wird dann an Eingang *labeled data* geliefert.

A.2.3 Performance

Der Unterprozess *Performance* wird nur ausgeführt, wenn an Eingang *test data* des *ActiveLearning*-Operators Daten geliefert wurden. Diese Daten werden dann in jeder Iteration mit dem *Apply*-Unterprozess klassifiziert und an den Ausgang *labeled data* des *Performance*-Unterprozesses geliefert. Dieser kann das Ergebnis dann bewerten und die Bewertung an Eingang *performance* liefern, damit sie in die *statistics*-Tabelle aufgenommen werden.

A.3 AverageLoop

Der Operator *AverageLoop* hat beliebig viele Eingänge, die an den Unterprozess weitergeleitet werden. Dieser kann den Operator *ActiveLearning* enthalten und die Ergebnis-Tabelle weitergeben. Der Unterprozess wird so oft aufgerufen, wie im Parameter *num iterations* spezifiziert. Die Ergebnis-Tabellen werden dann kombiniert, indem für alle Zellen der Durchschnitt berechnet wird.

Abbildungsverzeichnis

Literaturverzeichnis

[1] *KobRA (Korpus-basierte Recherche und Analyse mit Hilfe von Data-Mining).* `http://www.kobra.tu-dortmund.de`.

[2] Ali, Alnur, Rich Caruana und Ashish Kapoor: *Active Learning with Model Selection.* In: Brodley, Carla E. und Peter Stone (Herausgeber): *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, Seiten 1673–1679. AAAI Press, 2014.

[3] Attenberg, Josh und Foster Provost: *Why Label when You Can Search?: Alternatives to Active Learning for Applying Human Resources to Build Classification Models Under Extreme Class Imbalance.* In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, Seiten 423–432, New York, NY, USA, 2010. ACM.

[4] Balcan, Maria-Florina, Alina Beygelzimer und John Langford: *Agnostic Active Learning.* In: *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, Seiten 65–72, New York, NY, USA, 2006. ACM.

[5] Blitzer, John, Mark Dredze und Fernando Pereira: *Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification.* In: *ACL*, Band 7, Seiten 440–447. Citeseer, 2007.

[6] Breiman, Leo: *Bagging Predictors.* Machine Learning, 24(2):123–140, 1996.

[7] Brinker, Klaus: *Incorporating diversity in active learning with support vector machines.* In *In Proceedings of the 20th International Conference on Machine Learning*, pages 59–66. AAAI Press, 2003.

[8] Burges, Christopher J. C.: *A Tutorial on Support Vector Machines for Pattern Recognition.* Data Min. Knowl. Discov., 2(2):121–167, June 1998.

[9] Chang, Chih-Chung and Chih-Jen Lin: *LIBSVM: A library for support vector machines.* ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] Cohn, David, Richard Ladner and Alex Waibel: *Improving generalization with active learning.* In *Machine Learning*, pages 201–221, 1994.

[11] Fu, Yifan, Xingquan Zhu and Bin Li: *A survey on instance selection for active learning.* Knowledge and Information Systems, 35(2):249–283, 2013.

[12] Hsu, Wei-Ning and Hsuan-Tien Lin: *Active Learning by Learning.* 2015.

[13] Lodhi, Huma, Craig Saunders, John Shawe-Taylor, Nello Cristianini and Chris Watkins: *Text classification using string kernels.* The Journal of Machine Learning Research, 2:419–444, 2002.

[14] Mamitsuka, Naoki Abe Hiroshi: *Query Learning Strategies using Boosting and Bagging.*

[15] McCallum, Andrew and Kamal Nigam: *Employing EM and Pool-Based Active Learning for Text Classification.* In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 350–358, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[16] Mitchell, Tom M.: *Generalization as Search.* Artificial Intelligence, 18(2):203–226, 1982.

[17] Moschitti, Alessandro: *Making Tree Kernels Practical for Natural Language Learning.* In *EACL*, volume 113, page 24, 2006.

[18] Olsson, F.: *A literature survey of active machine learning in the context of natural language processing.* Technical Report 06, Box 1263, SE-164 29 Kista, Sweden, April 2009.

[19] Rapid-I: *RapidMiner.* https://rapidminer.com/.

[20] Rapid-I: *Text Mining Extension.* `http://marketplace.rapid-i.com/UpdateServer/faces/product_details.xhtml?productId=rmx_text`.

[21] Rapid-I: *How to extend Rapid Miner 5.* http://docs.rapid-i.com/r/whitepaper-how-to-extend, 2012.

[22] Settles, B.: *Active Learning.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2012.

[23] Settles, Burr: *Active Learning Literature Survey.* Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[24] Sharma, Manali and Mustafa Bilgic: *Most-Surely vs. Least-Surely Uncertain.* In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, 2013.

[25] Sheng, Victor S., Foster Provost and Panagiotis G. Ipeirotis: *Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers.* In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 614–622, New York, NY, USA, 2008. ACM.

[26] Tong, Simon and Daphne Koller: *Support Vector Machine Active Learning with Applications to Text Classification.* J. Mach. Learn. Res., 2:45–66, March 2002.

[27] Wallace, Byron C., Kevin Small, Carla E. Brodley and Thomas A. Trikalinos: *Who should label what? instance allocation in multiple expert active learning.* In *In Proc. of the SIAM International Conference on Data Mining (SDM*, 2011.

[28] Wu, Ting-Fan, Chih-Jen Lin and Ruby C Weng: *Probability estimates for multi-class classification by pairwise coupling.* The Journal of Machine Learning Research, 5:975–1005, 2004.

[29] Zhu, Jingbo, Huizhen Wang, Eduard Hovy and Matthew Ma: *Confidence-based Stopping Criteria for Active Learning for Data Annotation.* ACM Trans. Speech Lang. Process., 6(3):3:1–3:24, April 2010.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 30. Januar 2015

Tim Schendekehl