Ausarbeitung Projektgruppe Textmining

Nils Wortmann 2897970, Matthias Woelk 2630485

März 2020

Inhaltsverzeichnis

Zusammenfassung

In dieser Ausarbeitung beschäftigen wir uns mit der Kategorisierung von diversen Texten in unterschiedliche Kategorien. Dabei erklären wir zunächst die theoretischen Grundlagen der Support Vector Machine und des Hidden Markov Model. Außerdem erläutern wir nötige Schritte des Preprocessing um die Texte in das nötige Format zu bringen. Abschließend werden die beiden Ansätze evaluiert und miteinander verglichen. MW

1 Motivation

Heutzutage werden sehr viele Texte geschrieben und online gestellt. Es ist häufig von Interesse Inhalte aus diesen Texten zu ziehen, vor allem weil in der Wissenschaft immer mehr Texte unter freien Lizenzen im Internet landen und keiner mehr alle Paper seines Arbeitsbereiches lesen kann. Daher müssen unsere Computer ran und diese Paper analysieren und zumindest die wichtigsten Punkte herauszuarbeiten, wie zum Beispiel das Thema. Wir haben uns mit einem Bericht[1] beschäftigt, der beschreibt welche Methoden das sogenannte Text-Mining umfasst. Es gibt viele Anwendungsmöglichkeiten, wie z.B. Natural Language Processing, Information Extraction, Classification oder Clustering. Fast alle dieser Anwendungen haben gemein, dass diese nicht auf den rohen Textdaten arbeiten können. Daher muss man zunächst Preprocessing durchführen. Wir beschäftigen uns in dieser Arbeit mit den Schritten Tokenization und Stemming. Darauf folgend machen wir eine inhaltliche Analyse der Texte mithilfe von Support Vector Machine und Hidden Markov Model um eine Gruppe von Texte in vorgegeben Kategorien einzuordnen. NW

2 Datensätze

2.1 Twenty News Articles

20news ist eine Ansammlung von 20 sogenannten newsgroups. Diese sind Mailing Listen, die nach Themen sortiert sind. Es gibt 7 Hauptkategorien „alt“, comp“, „misc“, „rec“, „sci“, „soc“, und „talk“. Jede der 20 newsgroups hat ungefähr 800 Mails zu ihrem jeweiligen Thema. NW

2.2 Reuters Articles

Bei diesem Datenset handelt es sich um eine Sammlung von Reuters Artikeln. Insgesamt beinhaltet es circa 20.000 Artikel, von denen allerdings ungefähr die Hälfte fehlerhaft ist. Fehlerhaft heißt, dass die Kategorie fehlt, der Text nicht vorhanden oder nur aus Sonderzeichen besteht, oder das XML Markup fehlerhaft ist. Nach dem Preprocessing bleiben so noch 10378 Artikel übrig. Wir haben unsere Experimente hauptsächlich auf diesen Datensatz beschränkt. MW

3 Preprocessing

Beim Preprocessing ist das Ziel einen Text so vorzubereiten, dass unsere Modelle diese interpretieren können. Dafür nutzen wir zunächst Tokenization, welche die Worte aus dem Text heraus nimmt. Danach werden unnötige Worte eliminiert und jedes Wort auf seinen Stamm reduziert. NW

3.1 Tokenization

Für Tokenization wird als Eingabe ein Text als char Stream genommen und jedes Zeichen nacheinander durchgegangen. Dabei werden Tokens aus diesem Stream in ein Array gefüttert, das können Sätze oder Wörter sein, und unnötige Zeichen entfernt. Im Falle von Wörtern als Tokens wären das also Satzzeichen und Leerzeichen. Dabei muss man jedoch beachten, dass es manche Ausnahmen gibt, z.B. Abkürzungen, wo ein Punkt kein Satzzeichen mehr ist, oder Zahlen. Beim Beispiel Zahlen muss man sich entscheiden, wie diese interpretiert werden. Ob man sie zu Integern konvertiert oder, wie wir es getan haben, sie zu einem „/number/“Wort zusammenfasst.[2] NW

3.2 Stopwords

Nach der Tokenization müssen noch für die Kategorisierung potentiell störende Worte eliminieren. Diese sind Worte, die keinen Informationsgehalt tragen, wie Artikel, Pronomen und Fragewörter. Diese findet man in so gut wie jedem Text und tragen deshalb nicht inhaltlich dazu bei. Wir haben zur Eliminierung von Stopwords eine Liste benutzt.[3] NW

Wir haben diese Liste trotz der in 3.4 diskutierten Methode eingesetzt, da wir so die Möglichkeit haben, gezielt Wörter aus den Texten zu entfernen. Dies war zum Beispiel notwendig bei dem Wort „Reuters“, welches in vielen Artikeln des Reuters Datensets aus 2.2 enthalten war, aber nicht oft genug vorkam, um von den entsprechenden Schranken aus 3.4 abgefangen zu werden. MW

3.3 Stemming

In vielen Fällen ist es sinnvoll, Wörter mit ähnlichen Bedeutungen gleich zu behandeln. Die Worte „create“, „creation“und „created“sind inhaltlich sehr ähnlich. Wir brauchen also einen Weg, um diese Worte auf einen gemeinsamen Nenner zu reduzieren. M.F. Porter hat dazu einen Algorithmus für die englische Sprache entwickelt, der hier kurz erläutert wird. MW

Definition 3.1 *Sei ω ein Wort, welches den folgenden Aufbau hat:* $[C](VC)^m[V]$
Dabei gilt:
$[C]$*: Null oder mehr Konsonanten*
$[V]$*: Null oder mehr Vokale*
$(VC)^m$*: Einer oder mehr Vokale, gefolgt von einem oder mehr Konsonanten, m mal wiederholt*

Die Definition ist dem Artikel von Porter[4] entnommen. MW

Mithilfe dieser Definition eines Wortes und diverser Regeln, die Porter in seinem Artikel darlegt, ist es nun möglich, diverse Worte auf einen gemeinsamen Stamm zu reduzieren. Ein

Beispiel, welches Porter anführt ist folgendes:
generalizations → *generalization* → *generalize* → *general* → *gener*

Dabei ist gut zu erkennen, dass das Ergebnis des Algorithmus, nämlich „gener“nicht mehr unbedingt für einen Menschen erkennbar ist. Mit dieser Methode ist der Algorithmus in der Lage den Input Space um circa 30% zu reduzieren.[4] Wie Porter selbst anmerkt, arbeitet der Algorithmus außerdem nicht immer 100% genau. Es passieren also durchaus Fehler, auch wenn uns während unserer Tests keine aufgefallen sind. Womit der Algorithmus allerdings durchaus Probleme hat, sind Tippfehler, Rechtschreibfehler, oder sonstige Irregularitäten innerhalb einzelner Worte. Dies ist allerdings zu erwarten, da der Algorithmus diese nicht erkennen kann. Der Algorithmus interpretiert diese Worte folglich als neue Worte und verarbeitet sie als solche. MW

3.4 Reduktion des Input Space

Für die folgenden Definitionen nehmen wir an, dass es sich stets um Texte handelt, die bereits vollständig preprocessed sind.

Definition 3.2 *Sei* $\mathbb{A}$ *die Menge aller Worte eines Textes.*

Definition 3.3 *Es sei* $\mathbb{K}_i$ *die Menge aller Texte, welcher der Kategorie i angehören.*
$\omega \in \mathbb{K}_i$ *gibt dabei an, dass ein* ω *in mindestens einem der Artikel in* $\mathbb{K}_i$ *vorkommt.*

Definition 3.4 *Sei* Σ_ω *die Anzahl von* ω *über allen Texten.*
Sei weiterhin $\Sigma_\omega^{\mathbb{K}_i}$ *die Anzahl von* ω *über allen Texten der Kategorie i und* $\Sigma_\omega^{\mathbb{A}}$ *die Anzahl von* ω *im Text* $\mathbb{A}$*.*

Der letzte Schritt des Preprocessing besteht in der Eliminierung von Wörtern, die wenig bis keinen Mehrwert bei der Kategorisierung bieten, aber dennoch nicht zu den Stopwords gehören. Im Allgemeinen sind dies Wörter, die sehr wohl Informationsgehalt haben können, diesen aber aufgrund ihrer Häufigkeit verlieren. MW

Wenn ein Wort nur einmal in einem einzigen Text vorkommt, ist davon auszugehen, dass es für die Kategorisierung des Textes keinen Mehrwert bietet. Wäre ein solches Wort ein entscheidendes Merkmal der Kategorie eines Textes, ist davon auszugehen, dass es mehrfach vorkommt. Dies würde dazu führen, dass es nicht mehr von unserem Algorithmus entfernt wird. MW

Wir entfernen aus unseren Texten sowohl Wörter, die sehr häufig vorkommen (Highcut), als auch solche, die nur sehr selten vorkommen(Lowcut). Dazu zählen wir die Anzahl der einzelnen Worte über allen Texten nach dem Stemming und machen danach die Cuts. Dabei gelten die folgenden Grenzwerte:

MW

Highcut: $\Sigma_\omega > 10.000$
Lowcut: $\Sigma_\omega < 10$

Außerdem werden Wörter entfernt, welche in insgesamt weniger als zehn Artikeln vorkommen. Dies geschieht unabhängig von ihrer jeweiligen Häufigkeit innerhalb des jeweiligen Textes. Diese drei Einschränkungen werden vor allem dadurch nötig, das wir nach dem

Stemming einen Input Space von mehr als 20.000 unterschiedlichen Wörtern haben, was den verfügbaren Arbeitsspeicher überschreitet. Die einzelnen Zahlenwerte werden daraufhin schrittweise erhöht, bis ein Input Space mit einer Dimension von circa 4.600 unterschiedlichen Wörtern erreicht wurde. Vor allem die zweite Bedingung erfüllt außerdem den Zweck, dass sie Worte herausfiltert, die Tippfehler enthalten, oder so selten sind, dass sie keinen Mehrwert bieten. MW

Es sind also stets die folgenden Bedingungen erfüllt:

$$\forall \omega : 10 < \Sigma_\omega < 10.000$$

$$\forall \mathbb{A}, \omega : |\{\mathbb{A} | \omega \in \mathbb{A}\}| > 10$$

4 Kategorisierung

4.1 Support Vector Machine

Die Formeln in den Teilen 4.1.1 und 4.1.2 sind dem Artikel von Christopher Burges[5] entnommen und zum Zweck der Erklärung teilweise abgewandelt worden.

4.1.1 Grundlage

Die Support Vector Machine (SVM) beruht auf der Idee, mithilfe einer Hyperebene im n-dimensionalen Raum, Vektoren in zwei Kategorien zu unterteilen. Dabei wird versucht, den Abstand (Margin) der Hyperebene zu den Vektoren zu maximieren. Vektoren, die dabei den geringsten Abstand zur Hyperebene haben, also genau die Margin, nennt man Support Vector. MW

Formal versucht die SVM also, die folgende Menge von m Elementen möglichst eindeutig voneinander zu trennen. Dabei sei x_i das zu trainierende Element und y_i die entsprechende Kategorie des Elements.

$$\{(x_i, y_i) | i \in \{1, ..., m\} \wedge y_i \in \{-1, 1\}\}$$

Eine solche Hyperebene ist für einen beliebigen Normalvektor v eindeutig definiert als:

$$\langle x, v \rangle + b = 0$$

Wir beschreiben die Hyperbene folglich als eine Ebene, die senkrecht zu besagtem Normalvektor im Raum liegt[5]. Dabei bietet uns diese Definition den Vorteil, dass alle Punkte, die auf der Hyperebene liegen, diese Gleichung erfüllen. Liegen die Punkte nicht auf der Hyperebene, so gilt folgendes:

$$\langle x, v \rangle + b \begin{cases} > 0, & \text{wenn x auf der Seite ist in die } v \text{ zeigt} \\ < 0, & \text{sonst} \end{cases}$$

Wir haben also mit der Definition der Hyperbene auch gleichzeitig die Kategorisierung vorgenommen. Ist das Ergebnis größer 0 gehört x_i zur Kategorie $y_i = 1$, ist es kleiner, gehört es zur Anderen ($y_i = -1$). MW

Wenn wir nun annehmen, dass es eine solche Hyperebene gibt, was nicht immer der Fall ist, so können wir eine Hyperebene finden, die beliebig dicht an mindestens einem der x_i liegt. Haben wir nun mithilfe der SVM eine solche Ebene gefunden, dann stellt uns dies vor das Problem, dass um dieses Trainingsbeispiel möglicherweise weitere Elemente dieser Kategorie liegen, die wir bisher noch nicht trainiert haben. Das sorgt dafür, dass die SVM fehleranfällig für genau diese Elemente ist, da es sein kann, dass sie nach dem Training auf der falschen Seite liegen. Um diesen Fehlerfall zu minimieren, muss die Margin also maximiert werden. MW

Formal gesehen muss also folgendes Problem gelöst werden:

$$max(y_i(\langle v, x_i\rangle + b)) \text{ für } 1 \leq i \leq m$$

Das alleine reicht allerdings noch nicht aus, da wir nun einfach v unendlich groß wählen können. Dadurch wurde unser Maximum immer größer und wir hätten doch keine sinnvolle Hyperebene gefunden. Wir brauchen also noch eine Bedingung die v möglichst klein hält. Hierfür bietet sich die Euklidische Norm an, welche minimiert werden soll. Durch die Wahl von v und b kann die SVM nun also das folgende Problem lösen:

$$min\left(\frac{1}{2}||v||_2^2\right)$$
$$\text{Nebenbedingung:}$$
$$y_i(\langle v, x_i\rangle + b) \geq 1 \text{ für alle } 1 \leq i \leq m$$

Es wird also die Länge von v minimiert. Damit wird folglich die Margin maximiert und gleichzeitig eine untere Schranke für die Länge von v, abhängig von den Elementen die zu kategorisieren sind, gegeben. MW

4.1.2 Nicht linear separierbare Daten

Wie zuvor diskutiert, kann eine SVM genau zwei Kategorien unterscheiden.Dabei müssen die Daten zu den einzelnen Kategorien linear separierbar sein. Ist dies nicht der Fall, so haben $(\langle v, x_i\rangle + b)$ und y_i nicht mehr das gleiche Vorzeichen. Dies führt wiederum dazu, dass die Nebenbedingung aus 4.1.1 scheitert, da das Ergebnis in jedem Fall kleiner 1 ist. MW

Um nicht linear separierbare Daten kategorisieren zu können, ändern wir in der Folge unsere Formeln ab. Wir fügen eine Schlupfvariable ξ_i ein, welche dafür sorgt, dass unsere Nebenbedingung angepasst werden darf. Unsere neue Nebenbedingung sieht also folgendermaßen aus.

$$y_i(\langle v, x_i\rangle + b) \geq 1 - \xi_i \text{ für alle } 1 \leq i \leq m$$

Dadurch, dass wir ξ_i von 1 subtrahieren, können wir nun für jedes x festlegen, wie sehr die Nebenbedingung aus 4.1.1 verletzt werden darf. Zusätzlich wird aber auch hier wieder eine Schranke benötigt, da wir ξ_i ansonsten beliebig groß wählen könnten, um danach v auch wieder beliebig groß wählen zu können. Wir fügen also ξ_i zu unserem Minimierungsproblem

hinzu und erhalten folgende Gleichungen.

$$min\left(\frac{1}{2}||v||_2^2 + C\sum_{i=1}^{m}\xi_i\right)$$

Nebenbedingung:

$$y_i(\langle v, x_i\rangle + b) \geq 1 - \xi_i \text{ für alle } 1 \leq i \leq m$$

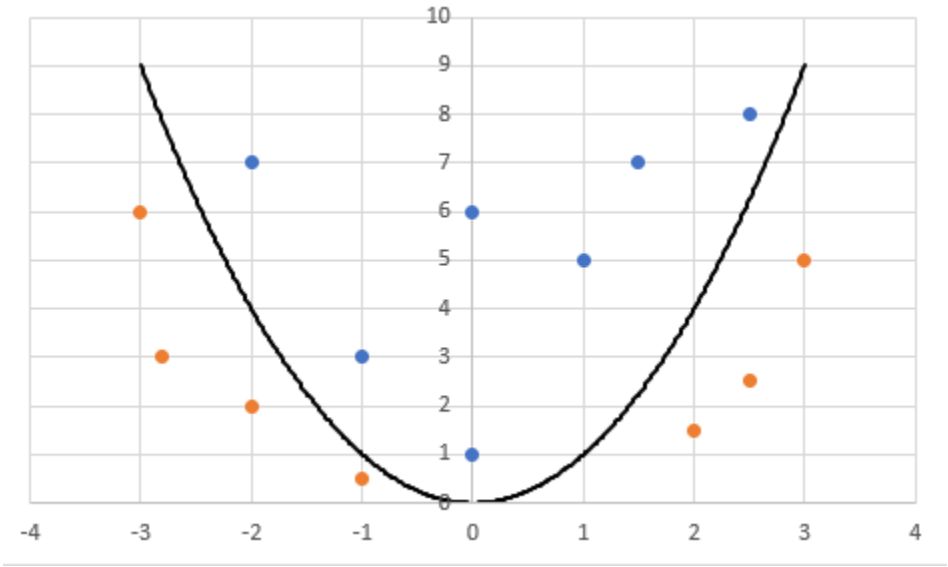
Der Parameter C ist dabei für die Regularisierung zuständig. Mit seiner Hilfe lässt sich angeben, wie stark die Summe der ξ_i in den zu minimierenden Wert mit einfließt. Dabei muss stets gelten $C > 0$.

$$C\begin{cases}\text{wenn } 0 < C < 1, & \text{Fehler werden weniger beachtet}\\ \text{sonst}, & \text{Fehler fallen mehr ins Gewicht}\end{cases}$$

Je kleiner also C ist, desto größere Fehler werden akzeptiert um v zu minimieren. Dies kann letztendlich dazu führen, dass übermäßig viele Elemente falsch kategorisiert werden, um im Austausch eine größere Margin zu erhalten. MW

4.1.3 Kernel

Im vorherigen Abschnitt wurde erklärt, wie man mit nicht-linearen Daten umgeht. In diesem Abschnitt wird nun geklärt, wie man aus dem nicht linear separierbaren Input Space einen linear separierbaren Feature Space generieren kann. Dazu sei das folgende Beispiel gegeben: MW



Wie in dem Bild gut zu sehen ist, sind die Daten nicht linear separierbar. Mit einer quadratischen Funktion, in diesem Fall der Normalparabel, ist dies jedoch problemlos möglich. Wir brauchen also eine Funktion, die wir auf alle Elemente anwenden können um danach einen linear separierbaren Feature Space zu erhalten, auf denen wir mit den Formeln aus 4.1.2 sinnvoll arbeiten können. Diese Funktion bezeichnen wir als Kernel K. Darüber hinaus stellen wir an die Kernelfunktion folgende Anforderungen:

$$K(x, x') = K(x', x) \tag{1}$$

$$K(x, x') \leq \sqrt{K(x, x)K(x', x')} \tag{2}$$

$$K(x, x) \geq 0 \tag{3}$$

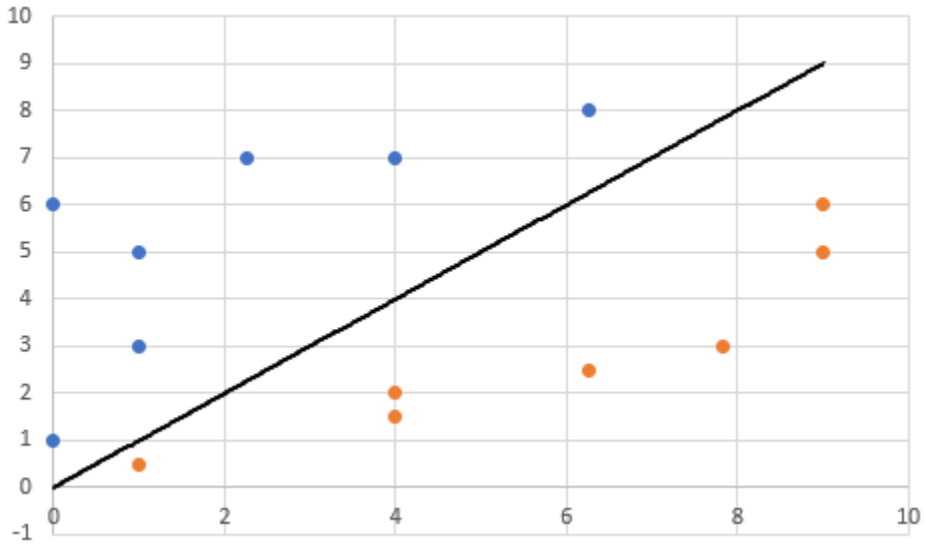
Wir geben also vor, dass unsere Kernelfunktion symmetrisch (1) sein soll. Außerdem muss der Satz von Cauchy Schwarz gelten (2) und die Funktion muss positiv semidefinit sein (3). MW

Dabei ist anzumerken, dass eine Kernelfunktion den Input Space auch in einen höherdimensionalen Feature Space überführen kann. Wie allerdings von Jin und Wang festgestellt, erhöht dies die Wahrscheinlichkeit von Generalisierungsfehlern. Dies kann allerdings mit genügend Trainingselementen kompensiert werden.[6] MW

Auf das vorherige Beispiel wenden wir also den folgenden Kernel an:

$$K(x) = (x_1^2, x_2)$$

In diesem Beispiel verwenden wir einen sehr simplen Kernel, der die x-Achse des 2-dimensionalen Input Space quadriert und so einen wiederum 2-dimensionalen Feature Space liefert.



Wie gut zu erkennen ist, sind die Daten nun linear separierbar und wir können mit den uns bekannten Formeln arbeiten, ohne dabei Fehler bei der Kategorisierung in Kauf nehmen zu müssen. Eine gut gewählte Kernelfunktion steigert also die Performance der SVM und ermöglicht es vorher nicht separierbare Probleme zu lösen. MW

4.1.4 Multi-Class SVM

Wie bereits beschrieben, unterscheidet eine SVM in genau zwei Kategorien ($y_i = 1 \vee y_i = -1$). Dies ist für die Kategorisierung von Texten jedoch in der Regel zu wenig. Zur Lösung dieses Problems bieten sich zwei Ansätze an: One vs One (OvO) und One vs Rest (OvR). MW

Wenn K die Anzahl der Kategorien ist, so werden beim OvO Ansatz genau $K(K-1)/2$ Modelle trainiert. Die Modelle unterscheiden dabei jeweils zwischen zwei der Kategorien. Das Ergebnis wird dann mit einem Majority Vote entschieden. Es wird also diejenige Kategorie gewählt, die die meisten Treffer erhält. MW

Beim OvR Ansatz werden genau K Modelle trainiert, die jeweils zwischen einer Kategorie und allen anderen unterscheiden. Am Ende wir dann die Kategorie gewählt, die den größten Konfidenzwert erzielt. Der Konfidenzwert ist dabei die positive oder negative Distanz zur Ebene. Die Distanz ist positiv, wenn die SVM das Element als zur Kategorie gehörig einstuft, und negativ, wenn dies nicht der Fall ist. MW

4.2 Hidden Markov Model

Ein Hidden Markov Model kann durch einen gerichteten Graphen beschrieben werden. Dabei gibt es zwei Arten von Zuständen, die die Knoten darstellen. Die einen sind versteckte Zustände, welche man nicht direkt beobachten kann, aber dennoch Einfluss auf die nächste Kategorie von Zuständen haben. Die anderen sind beobachtbare Zustände, in unserem Fall also die Worte aus denen ein Text besteht. Die Kanten sind Zustandsübergänge zum einen von jedem versteckten Zustand zu jedem anderen mit dem Kantengewicht der Übergangswahrscheinlichkeit und zum anderen gibt es Kanten, die die versteckten Zustände mit den beobachtbaren verbinden, ebenfalls mit Wahrscheinlichkeiten als Kantengewichte. Die formale Definition eines Hidden Markov Models sieht so aus:[7]
Ein Hidden Markov Model ist ein Fünf-Tupel (X, Y, A, B, Π)
N viele versteckte Zustände: $X = X_1, X_2, ..., X_N$
M mögliche Emissionen: $Y = Y_1, Y_2, ..., Y_M$
Wahrscheinlichkeiten der Zustandsübergänge:
$A = a_i j = P(q_{t+1} = X_j | q_t = X_i), 1 \leq i, j \leq N$
Wahrscheinlichkeiten der Beobachtungsübergänge:
$B = n_j(Y_k) = P(Y_k beit | q_t = X_j), 1 \leq j \leq N, 1 \leq k \leq M$
Wahrscheinlichkeiten des Startzustands:
$\Pi \pi_i = P(q_t = X_i), 1 \leq i \leq N$

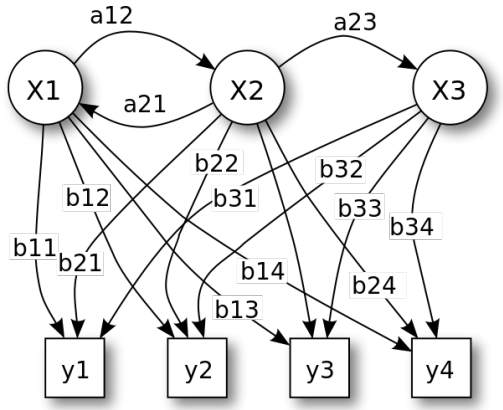


Abbildung 1: Beispiel für Hidden Markov Model mit $|X| = 3$ und $|Y| = 4$

Es gibt viele Möglichkeiten Hidden Markov Modelle zu implementieren. Ich habe es mit

dem Baum-Welch- und dem Viterbi-Algorithmus versucht. Diese werden im Folgenden vorgestellt. NW

4.2.1 Baum-Welch-Algorithmus

Der Baum-Welch-Algorithmus ist ein Algorithmus für semi-supervised learning von Hidden Markov Models. Man gibt dem Algorithmus die Struktur des Modells vor und er findet anhand von Beispielen eine lokal optimale Belegung der zuvor zufällig initialisierten Parameter. Grundlage des Algorithmus ist Expectation Maximization. Gegeben eine observierte Sequenz $O = (o_1, o_2, \ldots, o_T)$ werden von beiden Seiten der Sequenz der wahrscheinlichste Zustand für jeden Zeitpunkt berechnet. Von vorne wird $\alpha_i(t) = P(O_1 = o_1, ..., O_t = o_t, X_t = i|\theta)$ so definiert und rekursiv berechnet:
1. $\alpha(1) = \pi_i b_i(o_1)$
2. $\alpha_i(t+1) = b_i(o_{t+1}) * \sum_{j=1}^{N} \alpha_j(t) a_j i$
Von hinten wird $\beta_i(t) = P(O_{t+1} = o_{t+1}, ..., O_T = o_T|X_t = i, \theta)$ ebenfalls rekursiv berechnet:
1. $\beta_i(T) = 1$
2. $\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_i j b_j(o_{t+1})$

Mit diesen Funktionen lässt sich nun die Wahrscheinlichkeit, dass gegeben das Modell θ und Beobachtungssequenz O der zum Zeitpunkt t zugehörige Zustand X_i ist, berechnen:
$\gamma_i(t) = P(X_t = i|O, \theta) = \frac{P(X_t=i,O|\theta)]}{P(O|\theta)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$
Desweiteren sei $\xi_i j$ die Wahrscheinlichkeit, dass gegeben Modell θ und Sequenz O, der Zustand bei Zeit t X_i ist und bei t+1 X_j:
$\xi_i j(t) = P(X_t = i, X_{t+1} = j|O, \theta) = \frac{P(X_t, X_{t+1}=j, O|\theta)}{P(O|\theta)} = \frac{\alpha_i(t) a_i j \beta_j(t+1) b_j(o_{t+1})}{\sum_{k=1}^{N} \sum_{w=1}^{N} \alpha_k(t) a_k w \beta_w(t+1) b_w(o_{t+1})}$

Damit kann man nun die Parameter des Modells θ anpassen:
$\pi_i = \gamma_i(1)$
$a_i j = \frac{\sum_{t=1}^{T-1} \xi_i j(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$
$b_i(y_k) = \frac{\sum_{t=1}^{T} \mathbb{1}_{o_t=y_k} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)}$

NW

4.2.2 Viterbi-Algorithmus

Mit dem Viterbi-Algorithmus kann man einige statistische Probleme lösen, so auch die wahrscheinlichste Sequenz von versteckten Zuständen, die eine gegebene Emission erzeugen. Der Algorithmus generiert anhand des Modells θ und einer Eingabesequenz O einen Pfad durch die versteckten Zustände, welcher die höchste Wahrscheinlichkeit aus allen möglichen Pfaden hat. Der Algorithmus arbeitet mit dynamischer Programmierung und hat eine Laufzeit von $O(T^2 * K)$.[7]

NW

Algorithm 1
Viterbi-Algorithmus, Quelle Wikipedia[8]

function VITERBI(Y, X, Π, O, A, B): P
  for each state $i = 1, 2, ..., N$ **do**
    $T_1[i, 1] \leftarrow \pi_i \cdot b_i(o_1)$
    $T_2[i, 1] \leftarrow 0$
  end for
  for each observation $j = 2, 3, ..., T$ **do**
    for each state $i = 1, 2, ..., N$ **do**
      $T_i[i, j] \leftarrow \max_k(T_i[k, k-1] \cdot a_k i \cdot b_i(o_j))$
      $T_i[i, j] \leftarrow \arg\max_k(T_i[k, k-1] \cdot a_k i \cdot b_i(o_j))$
    end for
  end for
  $z_T \leftarrow \arg\max_k(T_1[k, T])$
  $p_T \leftarrow s_{z_T}$
  for $j = T, T-1, \ldots, 2$ **do**
    $z_{j-1} \leftarrow T_2[z_j, j]$
    $p_{j-1} \leftarrow s_{z_{j-1}}$
  end for
  return P
end function

5 Implementierung

Für die Implementierung unseres Projektes haben wir die Sprache Python gewählt, da sie gute Bibliotheken bereit stellt, und eine gute Basis für wissenschaftliche Arbeiten bietet.

Die Texte haben wir zunächst als String ausgelesen, welcher dann dem Tokenizer übergeben wurde. Anschließend wurden, wie eingangs beschrieben, die Stopwords entfernt, alle Zahlen gegen „/number/“getauscht und das Stemming durchgeführt. Anschließend wurde die in 3.4 beschriebene Reduktion vorgenommen. Das so entstandene Array mit den übrig gebliebenen Worten wurde sowohl der SVM, als auch dem HMM, zur Verfügung gestellt. MW

5.1 Support Vector Machine

5.1.1 Vector generation

Für die Implementierung der SVM wurde die Bibliothek Scikit-learn[9] verwendet, da wir bereits in früheren Projekten damit gearbeitet haben und sie eine umfassende Funktionalität bereitstellt.

Da die SVM mit Vektoren arbeitet, musste das Array aus Worten zunächst in einen solchen konvertiert werden. Dazu wurde zuvor eine Bibliothek mit allen Worten in allen Artikeln angelegt. Diese fungiert als Nullvektor, zu dem dann der jeweilige Artikelvektor hinzu addiert wird. Technisch wird das dadurch realisiert, dass Python einen Counter zur Verfügung stellt, der identische Einträge in einem Array zählt. Dabei besteht der Counter aus Tupeln c, die jeweils aus dem Wort w_c und der bisher gezählten Anzahl a_c bestehen. Für

jeden Text starten wir also mit einem Counter $\mathbb{C}$ für den gilt:

$$\forall c \in \mathbb{C} : a_c = 0$$

Nun können wir den Counter einfach die Anzahl der Worte des jeweiligen Textes zählen lassen.
Gegeben sei das Beispiel $[a, b, c, d, a]$ eines Textes nach dem Preprocessing und dem Counter $[(a, 0); (b, 0); (c, 0); (d, 0); (e, 0)]$.
Daraus ergibt sich nun nach dem Zählen folgender neuer Counter:
$[(a, 2); (b, 1); (c, 1); (d, 1); (e, 0)]$ MW

Nachdem wir also die Anzahl der Worte ermittelt haben, können wir den Counter in einen Vektor konvertieren. Dazu nehmen wir einfach die a_c der Tupel und fügen sie dem Vektor hinzu. Wichtig dabei ist, dass die Sequenz der Worte immer gleich bleibt, da die Vektoren sonst nicht vergleichbar sind. Schlussendlich wird der Vektor noch normalisiert, damit man ihn mit den Vektoren anderer Texte vergleichen kann. MW

5.1.2 Kernel

Als Kernel für die Kategorisierung der Datensets wurden 3 Kernelfunktionen in Erwägung gezogen. Ein linerarer Kernel (4), ein polynomieller Kernel (5) und ein Kernel mit radial basis function (6), welcher im weiteren Verlauf als rbf Kernel bezeichnet wird.

$$K(x, x') = \langle x, x' \rangle \tag{4}$$

$$K(x, x') = (\gamma \langle x, x' \rangle + r)^d \tag{5}$$

$$K(x, x') = exp\left(-\gamma ||x - x'||^2\right)^d \tag{6}$$

Die Parameter der Kernel sind also beim polynomiellen Kernel d, r und γ. Beim rbf Kernel steht nur γ zur Verfügung, welches außerdem positiv sein muss[10]. Der lineare Kernel bietet keine zusätzlichen Parameter. Für alle Kernel muss weiterhin der Parameter C aus 4.1.2 gewählt werden. MW

Um nun den besten Kernel für die Kategorisierung der Datensets zu finden, haben wir mithilfe von GridSearch alle Möglichkeiten durchprobiert. Die Parameter waren dabei wie folgt:

Kernel	C	γ	d	r
linear	{1;100;1000}	{}	{}	{}
poly	{1;100;1000}	{0,001; 0,005; 0,1; 1; 3; 5}	{1;2;3;4;5}	{0,1}
rbf	{1;100;1000}	{0,001; 0,005; 0,1; 1; 3; 5}	{}	{}

Tabelle 1: Parameter Grid für die SVM Kernels

In der folgenden Tabelle sind die jeweils besten Konfigurationen aufgelistet. Der Score entspricht dabei einer Zahl zwischen 0 und 1, wobei 1 der bestmögliche Score ist. Berechnet wird dieser Score von der Library durch einen Estimator, dessen genaue Arbeitsweise in der Dokumentation des Codes nicht zu finden war. Allerdings decken sich die Ergebnisse mit Tests, die mit der SVM im Anschluss durchgeführt wurden, sodass wir davon ausgehen,

dass der Estimator das gewichtete F1-Measure abschätzt. Gewichtet bedeutet hier, dass die Performance der einzelnen Kategorien prozentual anhand der Anzahl ihrer Elemente in die Gesamtwertung einfließen. MW

Kernel	C	γ	d	r	Score
linear	100	-	-	-	$\approx 0,887$
poly	1	3	1	0,1	$\approx 0,889$
rbf	1000	0,001	-	-	$\approx \mathbf{0,890}$

Tabelle 2: Die Parameterkonfiguration mit dem besten Score

Wie der Tabelle zu entnehmen ist, gibt es mit der entsprechenden Konfiguration keine signifikanten Unterschiede zwischen den einzelnen Kernelfunktionen. Dies liegt vor allem daran, dass sich der rbf Kernel bei sehr kleinem γ ähnlich dem linearen Kernel verhält.[11] Dies gilt, durch seine Parameterwahl, für den polynomiellen Kernel. Für unsere weiteren Experimente verwenden wir den rbf Kernel mit den Parametern, die in dieser Tabelle aufgeführt sind. MW

5.2 Hidden Markov Model

Zunächst habe ich versucht Hidden Markov Modelle mithilfe des Baum-Welch- und des Viterbi-Algorithmus zu implementieren. Dabei kam zunächst das Problem auf, dass der Baum-Welch-Algorithmus die maximale Rekursionstiefe erreicht wodurch ich den Algorithmus nicht mehr rekursiv sondern iterativ implementieren musste. Bei der iterativen Implementierung war dann das Problem, dass einzelnen Werte zu klein wurden um sie noch mit float32 darstellen zu können. Daher habe ich mich der seqlearn-Bibliothek[12] bedient um HMM trotzdem zur Kategorisierung nutzen zu können. Die seqlearn-Bibliothek benötigt die Informationen im CoNLL Format (Conference on Natural Language Learning). Dieses hat die einzelnen Worte eines Textes in jeweils einer Zeile und mit Leerzeichen getrennt sind in der gleichen Zeile Eigenschaften dieses Wortes angegeben, wie z.B. dass es groß geschrieben ist. NW

6 Evaluation

6.1 Preprocessing

6.1.1 Reduktion des Input Space

Die Reduktion des Input Space war in unserem Fall aufgrund von Einschränkungen in der Hardware notwendig. Experimente von Saif et al. haben allerdings gezeigt, dass mit der Eliminierung von Wörtern auch immer ein Verlust von Informationen einhergeht, selbst wenn es sich dabei nur um Stopwords handelt.[13] MW

Aus diesem Grunde haben wir versucht die Dimension des Input Space so groß wie möglich zu halten. Dennoch sind hier Probleme aufgetreten, da es Kategorien gibt, die weniger als zehn Artikel haben. Dies führt dazu, dass für die Kategorie potenziell relevante Worte gelöscht werden können. Im Falle von Wörtern, die nur in dieser Kategorie vorkommen, also ein entscheidendes Merkmal bilden, werden diese von unserem Algorithmus entfernt. Dies geschieht

außerdem unabhängig der entsprechenden Häufigkeit innerhalb der Texte. Somit schneiden die an sich schon unterrepräsentierten Kategorien tendenziell noch schlechter ab, als sie es ohnehin schon würden. Dies wird bei der Evaluation der einzelnen Modelle auch noch einmal erläutert. MW

Des weiteren ergibt sich aus der Art, wie in 3.1 mit Zahlen umgegangen wird das Problem, dass das entsprechende „/number/“Wort sehr häufig vorkommen. Wie man in Tabelle 3 sehen kann, ist das Wort „/number/“das mit Abstand häufigste Wort. MW

Wort	Absolute Häufigkeit
/number/	147.101
said	27.383
reut	10.342
year	8.430
will	6.421
billion	6.018
shar	5.796
compani	5.700
net	5.495
bank	5.180

Tabelle 3: Die häufigsten Worte nach Stemming, aber vor der Reduktion

Kategorie	Anzahl Artikel
earn	3761
acq	2186
money-fx	574
crude	483
grain	489
trade	441
ship	204
interest	263
sugar	145
gold	121

Tabelle 4: Die Kategorien mit den meisten Artikeln

Diese übermäßige Häufigkeit des Wortes „/number/“kommt vermutlich daher, dass die Kategorien der Artikel ein Ungleichgewicht haben. Wie man in Tabelle 4 sieht, haben viele der häufigsten zehn Kategorien mit Finanzen zu tun. „earn“, „money-fx“, sowie „trade“und „interest“lassen alle darauf schließen, dass sie tendenziell mehr Zahlen enthalten, als Artikel mit vergleichbarer Länge einer anderen Kategorie. Dies müsste aber durch weitere Tests untermauert werden und ist an dieser Stelle Spekulation. Dafür spricht allerdings auch, dass das Wort „billion“sehr häufig vorkommt, was im Kontext das Vorkommen von Zahlen vermuten lässt. All diese Umstände führen dazu, dass Zahlen bei unserer Kategorisierung keine Rolle spielen, da sie aufgrund ihrer Häufigkeit vom Preprocessing vorab abgefangen werden. MW

6.2 Metrik

Für die Evaluation der beiden Modelle verwenden wir jeweils Accuracy, Precision, Recall und F-Measure. Davon sind Precision, Recall und F-Measure jeweils einmal gewichtet und einmal ungewichtet. Gewichtet bedeutet, dass die Scores der jeweiligen Kategorien prozentual bezogen auf die Anzahl ihrer Elemente in die Wertung mit eingehen.
Sei S_g der Score über allen Kategorien und S_k der Score für die jeweilige Kategorie. Sei weiterhin $\mathbb{K}$ die Menge aller Kategorien, $\mathbb{T}_k$ die Menge aller Texte einer Kategorie und $\mathbb{G}$ die Menge aller Texte, so gilt:

$$S_g = \begin{cases} \text{gewichtet}, & \sum \left(S_k * \frac{|\mathbb{T}_k|}{|\mathbb{G}|} \right) \\ \text{ungewichtet}, & \frac{\sum S_k}{|\mathbb{K}|} \end{cases}$$

MW

6.3 Support Vector Machine

Die Performance der SVM weist in unseren Experimenten teils gravierende Unterschiede in der Qualität auf. Wie man Tabelle 5 entnehmen kann, ist die ungewichtete Performance der SVM suboptimal.

Metrik	Ungewichtet	Gewichtet
Accuracy	84,48%	-
Precision	51,24%	82,84%
Recall	37,9%	22,95%
F-Measure	43,57%	23,98%

Tabelle 5: Performance SVM über allen Kategorien

Wie bereits zuvor erwähnt, und in Tabelle 4 gut zu erkennen, besteht ein massives Ungleichgewicht zwischen den Kategorien. Wir vermuten daher, dass die schlechte Performance der SVM von diesem Ungleichgewicht herrührt. MW

In den Reuters Artikeln gibt es insgesamt 66 Kategorien. 40 dieser Kategorien haben unter 50 Artikel. 19 dieser vergleichsweise sehr kleinen Kategorien haben darüber hinaus unter 20 Artikel. Wenn man diese Zahlen mit der Größe der häufigsten 10 Kategorien (Tabelle 4) vergleicht, so wird klar, dass diese 40 Kategorien keinen großen Einfluss auf die Hyperbene haben können. Die Schlupfvariable ξ aus 4.1.2 wird nicht groß genug, um eine Rolle beim Minimierungsproblem zu spielen. MW

Bestätigt wird diese Annahme durch die gewichteten Scores aus Tabelle 5. Da diese deutlich höher sind, ist anzunehmen, dass die SVM viele Texte aus den kleineren Kategorien nicht richtig kategorisiert hat. Um das zu überprüfen, haben wir uns entschieden zusätzlich einen Test mit den Kategorien durch zu führen, die mehr als 200 Artikel haben. MW

Metrik	Ungewichtet	Gewichtet
Accuracy	93,80%	-
Precision	89,34%	93,78%
Recall	86,07%	93,80%
F-Measure	87,67%	93,79%

Tabelle 6: Performance SVM über Kategorien mit mehr als 200 Artikeln

Wie man Tabelle 6 entnehmen kann, sind die Scores der SVM nun deutlich besser. Da es nun jedoch nur 8 Kategorien zu unterscheiden gibt, ist nicht auszuschließen, dass sich der Score auch deshalb verbessert hat, weil die Wahrscheinlichkeit für Fehler nun geringer ist. Ob dem so ist, müssten weitere Tests zeigen. Ein solcher Test wäre zum Beispiel, die zu kleinen Kategorien mit Duplikaten auf eine entsprechende Größe zu bringen. Damit wäre ausgeschlossen, dass die SVM die Texte bei weniger Kategorien zufällig richtig kategorisiert, da die Anzahl an Kategorien gleich bleibt. MW

Kategorien	C	γ
7 größten	1000	0,001
66	1000	0,1

Tabelle 7: Parameter für rbf Kernel

Auffallend ist außerdem, dass die GridSearch für die 7 größten Kategorien einen anderen γ Parameter gewählt hat als für alle Kategorien. Der Grund liegt vermutlich in der verringerten Anzahl an Kategorien. Ein größeres γ sorgt dafür, dass die einzelnen Elemente weniger Einfluss auf umliegende Elemente haben.[11] Dies führt letztendlich dazu, dass die SVM die Kategorien nicht so stark verallgemeinert. Da nun weniger Kategorien vorhanden sind, und diese auch von der Anzahl her dichter beieinander liegen als vorher, ist es möglich, diese besser voneinander zu trennen, was in dem größeren γ resultiert. MW

Abschließend kann man sagen, dass eine SVM zur Kategorisierung von Text gut geeignet ist. Dies wird auch von den Erkenntnissen von Joachims gestützt, welcher ebenfalls Texte mithilfe einer SVM kategorisiert hat.[14] Wie Yang und Liu in ihrem Artikel jedoch anmerken, fehlen in Joachims Artikel Daten und seine Vergleichsalgorithmen schneiden unerwartet schlecht ab. Obwohl die Ergebnisse der von Yang und Liu durchgeführten Experimente wesentlich dichter beeinander liegen, kommen sie dennoch zu dem Schluss, dass die SVM unter den getesteten Methoden zu den Besten zur Kategorisierung von Text gehört.[15] MW

6.4 Hidden Markov Model

Aus der Form des seqlearn-hmm werden vom Modell selber zunächst jedem einzelnen Wort eine Kategorie zugeordnet. Um Texte besser vergleichen zu können habe ich mich entschieden nur eine bestimmte Anzahl der häufigsten Worte, nach Häufigkeit sortiert, an das Modell zu geben. Zunächst waren es die 10 häufigsten Worte auf allen Kategorien und Zuordnung der Kategorie zu jedem einzelnen Wort:

Metrik	Ungewichtet	Gewichtet
Accuracy	22,95%	-
Precision	1,15%	25,12%
Recall	1,39%	22,95%
F-Measure	1,26%	23,98%

Tabelle 8: HMM für die 10 häufigsten Worte, für jedes einzelne Wort, alle Kategorien

Metrik	Ungewichtet	Gewichtet
Accuracy	22,13%	-
Precision	1,01%	21,86%
Recall	1,36%	22,13%
F-Measure	1,16%	21,99%

Tabelle 9: HMM für die 10 häufigsten Worte, für den ganzen Text, alle Kategorien

In Tabelle 8 sieht man, dass Hidden Markov Model im Durchschnitt sehr schlecht abschneidet und selbst gewichtet nur weniger als 24% F-Measure erreicht. Man sieht, dass es in Precision, Recall und F-Measure im Durchschnitt bei ca. 1% liegt und Gesamt steigt es jeweils auf etwas mehr als 20%. Daran sieht man, dass das HMM nicht sehr gut mit unbalanzierten Daten umgeht und dazu auch noch nicht gut kategorisiert. In Tabelle 9 sind die Metriken dargestellt, nachdem über Mehrheitsentscheid der Kategorien der Worte, die pro Text an das HMM gegeben wurden, eine Kategorie dem Text zugeordnet wurde. Auch hier scheint das Modell nicht gut zu Klassifizieren. Da das HMM scheinbar nicht gut funktioniert, weil die Klassen nicht ausgewogen sind habe ich wie bei der SVM nur die 6 häufigsten Kategorien als Datensatz benutzt. NW

Metrik	Ungewichtet	Gewichtet
Accuracy	29,8%	-
Precision	13,59%	35,33%
Recall	14,71%	29,80%
F-Measure	14,13%	32,33%

Tabelle 10: HMM für die 10 häufigsten Worte, für jedes einzelne Wort, häufigsten 6 Kategorien

Metrik	Ungewichtet	Gewichtet
Accuracy	28,59%	-
Precision	11,41%	29,12%
Recall	14,27%	28,59%
F-Measure	12,68%	28,85%

Tabelle 11: HMM für die 10 häufigsten Worte, für jedes einzelne Wort, häufigsten 6 Kategorien

Wie man in Tabelle 10 sieht, ist das Modell zwar deutlich besser im Durchschnitt aber schlechter als Raten. Dies ändert sich bei Kategorisierung für den gesamten Text wie in Tabelle 11 gezeigt nicht. NW

Die HMM ist offenbar nicht für Kategorisierung geeignet. Es wäre noch möglich den gesamten Vokabularvektor einzugeben, wodurch jedoch die Laufzeit sich deutlich verschlechtern würde oder man gibt dem HMM die Stützvektoren der SVM um sie zu trainieren und schaut, ob das besser funktioniert. NW

7 Schlussfolgerung

Wir haben in dieser Arbeit anhand des Reuters Datensatzes getestet, wie gut die Modelle Support Vector Machines und Hidden Markov Model Texte anhand der darin enthaltenen

Worte kategorisieren können. Dabei sind wir zu dem Ergebnis gekommen, dass Support Vector Machines sich gut eignen, jedoch darauf angewiesen sind, dass die Kategorien gleichmäßig verteilt sind, um alle Kategorien gut erkennen zu können. Hidden Markov Model eignen sich nicht gut dafür Texte zu kategorisieren, weil sie sehr Kontext abhängig sind und um den kompletten Text von einem Hidden Markov Model als Kontext zu haben bräuchte man riesige Modelle, was uns technisch nicht möglich ist und den Rahmen dieser Aufgabe sprengt. NW

8 Kennzeichnung Autor

Jeder Absatz ist mit den Initialen des Autors gekennzeichnet. Die Initialen sind jeweils am Ende des Absatzes und beziehen sich auf den kompletten vorherigen Absatz. Ein Absatz kann Tebellen und/oder Bilder enthalten, und beginnt immer mit einer Überschrift, oder einem eingerückten Text.

Literatur

[1] Mehdi Allahyari, Seyed Amin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *CoRR*, abs/1707.02919, 2017. URL: `http://arxiv.org/abs/1707.02919`, http://arxiv.org/abs/1707.02919 `arXiv:1707.02919`.

[2] Tanu Verma, Renu Renu, and Deepti Gaur. Tokenization and filtering process in rapidminer. *International Journal of Applied Information Systems*, 7:16–18, 04 2014. https://doi.org/10.5120/ijais14-451139 `doi:10.5120/ijais14-451139`.

[3] Damian Doyle. Stopwords, https://www.ranks.nl/stopwords. (accessed: 22.03.2020). URL: `https://www.ranks.nl/stopwords`.

[4] MF Porter. An algorithm for suffix stripping. *Program: Electronic Library and Information Systems*, 14, 03 1980. https://doi.org/10.1108/eb046814 `doi:10.1108/eb046814`.

[5] Christopher Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 01 1998. https://doi.org/10.1023/A:1009715923555 `doi:10.1023/A:1009715923555`.

[6] C. Jin and L. Wang. Dimensionality dependent pac-bayes margin bound. *Advances in Neural Information Processing Systems*, 2:1034–1042, 01 2012.

[7] R. Li, J. Zheng, and C. Pei. Text information extraction based on genetic algorithm and hidden markov model. In *2009 First International Workshop on Education Technology and Computer Science*, volume 1, pages 334–338, 2009.

[8] the free encyclopedia Wikipedia. Viterbi algorithm. (accessed: 27.03.2020). URL: `https://en.wikipedia.org/wiki/Viterbi_algorithm`.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] Scikit Learn. Overview svm kernel. (accessed: 25.03.2020). URL: `https://scikit-learn.org/stable/modules/svm.html`.

[11] Scikit Learn. Rbf svm parameters. (accessed: 25.03.2020). URL: `https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html`.

[12] Lars Buitinck. Seqlearn bibliothek. (accessed: 27.03.2020). URL: `http://larsmans.github.io/seqlearn/reference.html`.

[13] Hassan Saif, Miriam Fernandez, and Harith Alani. On stopwords, filtering and data sparsity for sentiment analysis of twitter. *Proceedings of the 9th International Language Resources and Evaluation Conference (LREC'14)*, pages 810–817, 01 2014.

[14] Thorsten Joachims. Text categorization with support vector machines. *Proc. European Conf. Machine Learning (ECML'98)*, 01 1998. https://doi.org/10.17877/DE290R-5097 `doi:10.17877/DE290R-5097`.

[15] Yiming Yang and Xin Liu. A re-examination of text categorization methods. *Proceedings of the 22nd SIGIR, New York, NY, USA*, 01 2003. https://doi.org/10.1145/312624.312647 `doi:10.1145/312624.312647`.