Westfälische Wilhelms-Universität Münster Institut für Informatik

Diplomarbeit

Detexify: Erkennung handgemalter LaTeX-Symbole

Daniel Kirsch

27. Oktober 2010

Betreut durch Prof. Dr. Xiaoyi Jiang

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Münster, 27. Oktober 2010

Inhaltsverzeichnis

Vorwort

Wie es zu dieser Diplomarbeit kam...

Eines Tages saß ich mit einem Kommilitonen zusammen in der Mensa. Wir unterhielten uns über Ideen im Allgemeinen und welche es sich lohne umzusetzen. Wir hatten viele Ideen und meistens keine Zeit sie zu verwirklichen. Tatsächlich existierte sogar eine Liste mit einigen davon - ein schlichtes Blatt Papier, auf dem sich über die Zeit einiges angesammelt hatte und wir nannten es den Ideenfriedhof.

Am besagten Tag ging es vor allem darum, wie man auf *gute* Ideen käme. Denn auf dem Ideenfriedhof ruhte Sinnvolles und weniger Sinnvolles. Ich äußerte die Behauptung, dass Ideen, die eigene Probleme lösen, solche seien, deren Umsetzung sich lohne, weil man in der Regel mit seinen Problemen nicht alleine dastehe. Somit löse man auch Probleme für andere.

Wir kamen auf andere Themen und da mein Kommilitone zu jener Zeit den Sekretärinnen des Mathematischen Instituts einen LaTeX-Kurs gab, unterhielten wir uns auch darüber. Irgendwann bemerkte er: „Da fällt mir etwas ein... Was fehlt, ist eine LaTeX-Rückwärts-Suche. Es kommt ganz häufig vor, dass man zwar weiß, was man für ein Symbol haben will, aber nicht den Befehl kennt. Das Problem hatten meine Sekretärinnen jetzt auch schon ein paar Mal. Wenn man jetzt ein Programm hätte, in das man das Symbol malen könnte und das dann den richtigen Befehl ausspuckt...“

Ich fand, dass das eine außergewöhnlich gute Idee war. Ich hatte das Problem zwar nicht selbst, aber ich würde es bald haben, wenn ich meine Diplomarbeit schreiben würde, denn ich würde sie in LaTeX schreiben. Ich hatte allerdings zu dem Zeitpunkt noch keine Ahnung, was das Thema meiner Diplomarbeit sein würde. Ich hatte zu diesem Zeitpunkt auch keine Berührung mit Mustererkennung irgendeiner Art gehabt und keine Ahnung von LaTeX. Ich fand aber das Problem so interessant, dass ich fest entschlossen war es anzugehen.

Einige Wochen später, nach vielem Lesen und Probieren, hatte ich eine benutzbare Version, implementiert als Website und bedienbar über einen Browser. Seitdem wird die Anwendung täglich von fast 1000 Besuchern genutzt. Erst später wurde mir klar, dass sich das Thema gut für eine Diplomarbeit eignet und zu meiner Freude war Prof. Jiang auch dieser Meinung. Zudem würde ich meine eigene Diplomarbeit bei der Erstellung meiner Diplomarbeit verwenden können. Wer kann das schon von sich behaupten?

Was ich aber eigentlich hier sagen will: Sie lesen nun im Wesentlichen das Ergebnis eines LaTeX-Kurses für Sekretärinnen des Mathematischen Institutes der WWU Münster. Darum gilt den damaligen Teilnehmerinnen mein besonderer Dank.

Für Julia Fiege, Yu-Mei Kao, Angela Odermann & Tamara Tietmeyer.

1. Motivation

1.1. LaTeX

Wenn es um das Verfassen wissenschaftlicher Texte geht, ist für viele LaTeX die erste Wahl. Anders als bei einem What You See Is What You Get (WYSIWYG)-Editor kümmert man sich nicht um das Layout, um Abstände und Schriftgrößen sondern um die Semantik des Geschriebenen. Man zeichnet die Kapitel, Abschnitte, Formeln usw. aus, eher wie in Hyper Text Markup Language (HTML) als in Textverarbeitungen. Der Vorteil liegt auf der Hand. Inhalt und Präsentation sind klar getrennt, was zur Folge hat, dass man nicht vom Wesentlichen abgelenkt wird, wenn man an Texten und Formeln arbeitet. Das Aussehen des Dokuments wird zentral in der Präambel definiert.[1]

LaTeX hat jedoch auch seine Nachteile. Gerade Anfänger haben es aufgrund der anfangs flachen Lernkurve schwer. Da man LaTeX in der Regel im Quelltext bearbeitet, also jeden Befehl von Hand schreibt, muss man sich unheimlich viele Befehle merken. Das beginnt bei einfachen Befehlen wie `\chapter`, deren Name sich geradezu aufdrängt, aber wenn es darum geht mathematische Formeln wie

$$\Gamma(x) = \int_0^\infty s^{x-1} e^{-s}\, ds$$

in das Dokument zu bringen, wird es schon schwieriger.

Dass man ein Γ mit dem Befehl `\Gamma` bekommt, ist noch zu erahnen. Bei dem Versuch ∞ durch `\infinity` zu erhalten, hätte man jedoch einen Fehler geerntet - der korrekte Befehl lautet `\infty`. Bei Befehlen wie `\leftrightsquigarrow` ($\leftrightsquigarrow$) hört dann jede Intuition auf. Es ist also offensichtlich, dass es einigen Raum für unterstützende Maßnahmen bei der Erstellung von LaTeX-Dokumenten gibt. Das gilt insbesondere für mathematische Formeln, in denen viele unterschiedliche Symbole vorkommen können.

1.2. Die Suche nach der optimalen Eingabemethode

Überlegt man sich die natürlichste oder zumindest gewohnteste Art Text oder Mathematik zu notieren, so kommt man unweigerlich auf Stift und Papier. Während bei Texten die Eingabe über eine Computertastatur durchaus schneller sein kann als das Schreiben mit

[1] Es gibt noch sehr viele weitere Vorteile, aber alle aufzuzählen würde hier den Rahmen sprengen.

einem Stift, ist spätestens bei Formeln klar, dass hier der Stift seine Vorzüge hat. Es können beliebige Formen, Zeichen und Symbole in beliebige räumliche Beziehung gebracht werden. Nichts liegt also näher, als diese Eingabeform in die digitale Welt übertragen zu wollen. Allgemein ist die Eingabe mit einer Tastatur bei kleinen Alphabeten schneller, jedoch bei großen Alphabeten nicht mehr praktikabel [45].

Stellt man sich also einen optimalen LaTeX-Editor vor, so würde er eine Fläche zur Verfügung stellen, auf die einfach mit Hilfe eines Grafiktabletts geschrieben und skizziert werden kann. Die Kurven und Linien würden dann vom Editor in Text, Tabellen, Formeln und Diagramme überführt – alles genau wie vom Benutzer erwartet. Dies ist eine Vision, die es seit 1995 gibt [29]. Leider sind wir 15 Jahre später immer noch nicht so weit.

Alleine der Bereich der mathematischen Formelerkennung ist noch nicht auf einem Level, auf dem man die verfügbaren Lösungen als ausgereift bezeichnen könnte. Es gibt kaum kommerzielle Lösungen und auch nur eine Hand voll experimentelle Tools. Einige davon haben auch LaTeX oder Textsatz allgemein als Fokus. Andere gehen eher in Richtung interaktive Mathematik oder sogar Computer Algebra Systems (CASs).

Im Folgenden werden einige der verfügbaren Lösungen vorgestellt und bewertet.

1.2.1. FFES - Freehand Formula Entry System

Smithies u. a. [38] beschreiben ein System, mit dem handgeschriebene Formeln in LaTeX, Mathematica oder eine LISP-artige Notation überführt werden können. Das System wird als Prototyp beschrieben. Leider habe ich das Projekt, dessen Quellcode frei verfügbar ist, nicht zum Laufen bringen können und mir somit keinen eigenen Eindruck verschaffen können. Die Autoren gehen in ihrem Artikel auf die Problematik von Erkennungsfehlern unterschiedlicher Art ein und geben an, wie sie diese behandeln. Zum Beispiel stellen sie Gesten bereit, mit denen eine falsche Gruppierung von Strichen korrigiert werden kann. Symbolerkennungsfehler können mithilfe eines Kontextmenüs, das die wahrscheinlichsten Symbole anzeigt, korrigiert werden. Den Autoren nach stellen Fehler beim Parsen der Formel aufgrund von geringfügig deplatzierten Strichen das größte Problem dar. Diese müssen dann durch manuelles Verschieben von Strichen korrigiert werden. Hier besteht die Gefahr in eine frustrierende Formel-Debugging-Schleife zu geraten, da es häufig unklar ist, an welchem Strich der Algorithmus genau scheitert. Abbildung 1.1 zeigt das Benutzerinterface von FFES. Der Screenshot ist der Internetseite [54] entnommen, wo auch der Quellcode des Projektes erhältlich ist.

Die Autoren bemerken zudem, dass ihr System insbesondere für Experten nicht schneller zu bedienen ist als klassische Systeme wie der Microsoft Equation Editor oder LaTeX, aber es sei komfortabler und leichter zu lernen.

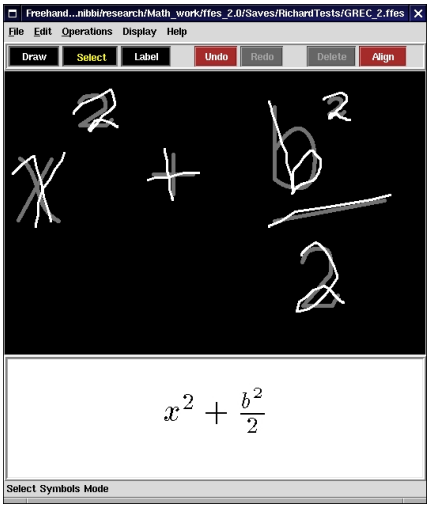


Abbildung 1.1.: Freehand Formula Entry System

1.2.2. Natural Log

Natural Log ist von seinen Zielen her ähnlich wie FFES einzustufen. Das als Java-Applet implementierte Programm wandelt handgeschriebene Formeln in LaTeX oder MathML um. Es kann unter [28] getestet werden. Als einzige Korrekturhilfe bietet es die Möglichkeit Striche durch eine Geste wieder zu löschen. Leider ist Natural Log durch die fehlenden Korrekturmöglichkeiten überhaupt nicht zu gebrauchen. Die Verwendung ist eine Qual. Abbildung 1.2 zeigt die Benutzeroberfläche von Natural Log.

1.2.3. JMathNotes

JMathNotes ist ein weiteres Programm zur Konvertierung von handgeschriebenen Formeln in LaTeX oder Mathematica. Es ist wie Natural Log in Java implementiert, wird aber nicht als Applet zur Verfügung gestellt, sondern ist als Desktopanwendung unter Linux und Windows installierbar. Es bietet dabei ähnliche Korrekturmöglichkeiten wie FFES. Die Gruppierung von Strichen kann korrigiert werden, einzelne Striche und ganze Symbole können verschoben werden und die Labels von falsch erkannten Symbolen können korrigiert werden. Für Letzteres muss man allerdings das richtige Label kennen. Man bekommt nicht wie in FFES eine Auswahl der besten Treffer (siehe 1.2.1). Im Wesentlichen ist das JMathNotes von denselben Problemen geplagt wie FFES, obwohl es fünf Jahre später veröffentlicht wurde (FFES ist aus 1999 und JMathNotes aus 2004). Abbildung 1.3 zeigt die Benutzeroberfläche von JMathNotes.

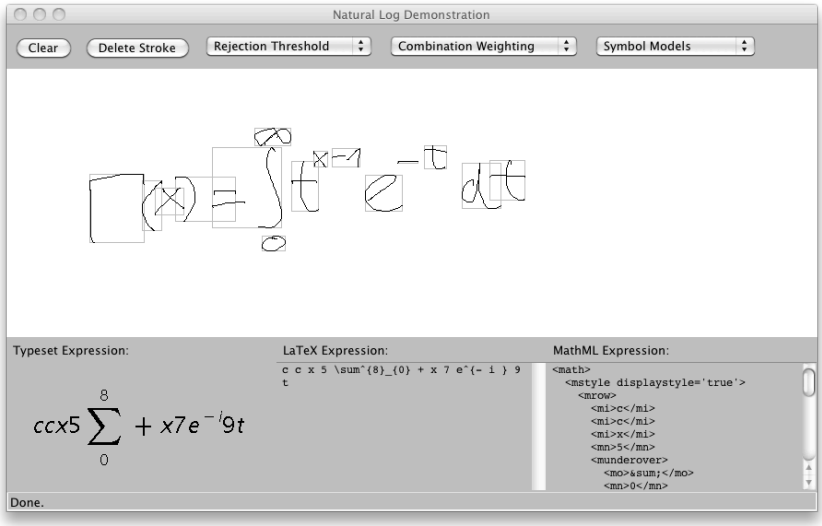


Abbildung 1.2.: Natural Log

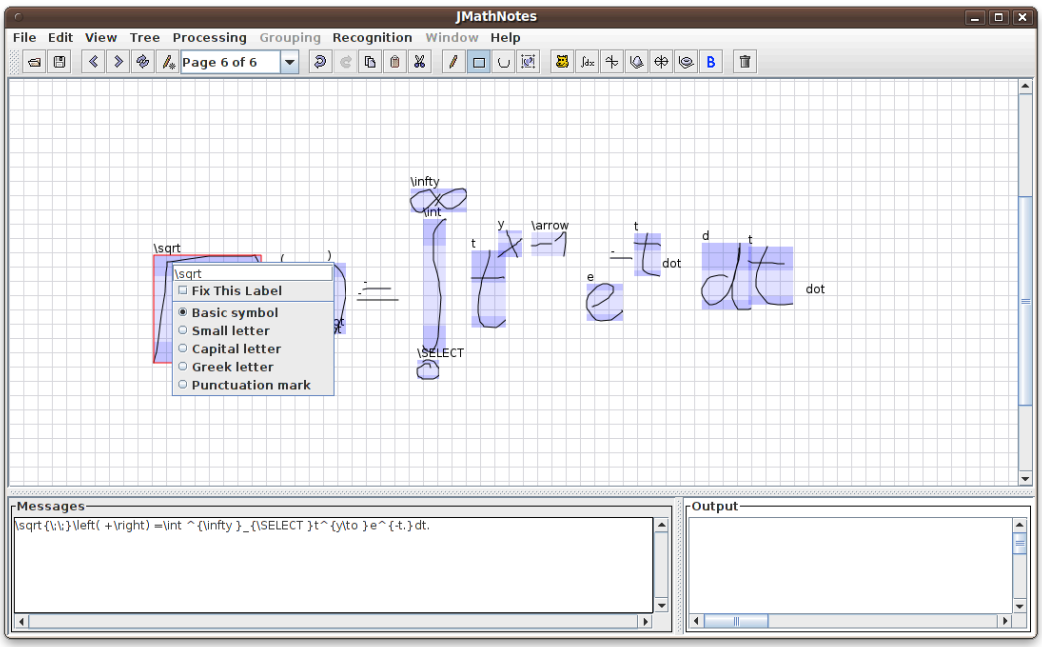


Abbildung 1.3.: JMathNotes

1.2.4. InftyEditor

Ein weiteres Beispiel für einen Editor, der beim Textsatz helfen soll, ist die kostenlose Software InftyEditor [39]. Sie ermöglicht die Eingabe von Formeln in einer Mischung aus WYSIWYG-Editor und LaTeX-Autovervollständigung mit Vorschau (siehe Abb. 1.4).

InftyEditor bietet außerdem die Funktion handgeschriebene Formeln zu erkennen. Dazu öffnet man das InftyHandWriting Input Pad und fängt an zu zeichnen. Die Erkennung erfolgt inkrementell, also immer nach einem kurzen Zeitintervall nachdem der letzte Strich gezeichnet wurde. Die Korrekturmöglichkeiten beschränken sich darauf, durch einen Klick auf Striche einige Möglichkeiten, wie diese zu interpretieren sind, durchzugehen. Einfache Formeln werden damit auch noch einigermaßen sicher erkannt. Sobald aber die Komplexität der Formel steigt, und vor allem sobald man der Software unbekannte Symbole braucht, wird die Benutzung zum Frusterlebnis. Abb. 1.5 zeigt das InftyHandwriting Input Pad.

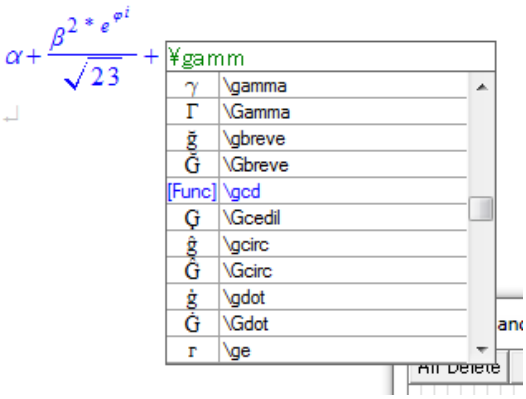


Abbildung 1.4.: InftyEditor Autovervollständigung

1.2.5. Microsoft Math

Microsoft Math ist ein kommerzielles Produkt, das aber nicht für den Textsatz gedacht ist, sondern eher mit einem programmierbaren Taschenrechner zu vergleichen ist. Die Eingabe erfolgt entweder über die Tatstatur (Abb. 1.6) oder über Handschrifterkennung (Abb. 1.7). Die Handschrifterkennung macht zwar einen solideren Eindruck als bei den vorgenannten Produkten, weist aber ebenso noch Verbesserungspotential auf. Abbildung 1.7 zeigt einen Versuch die Gammafunktion zu erkennen.

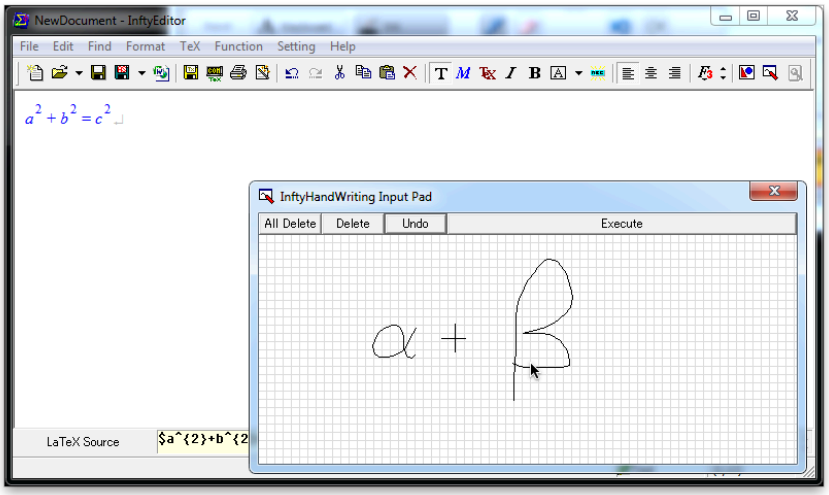


Abbildung 1.5.: InftyEditor

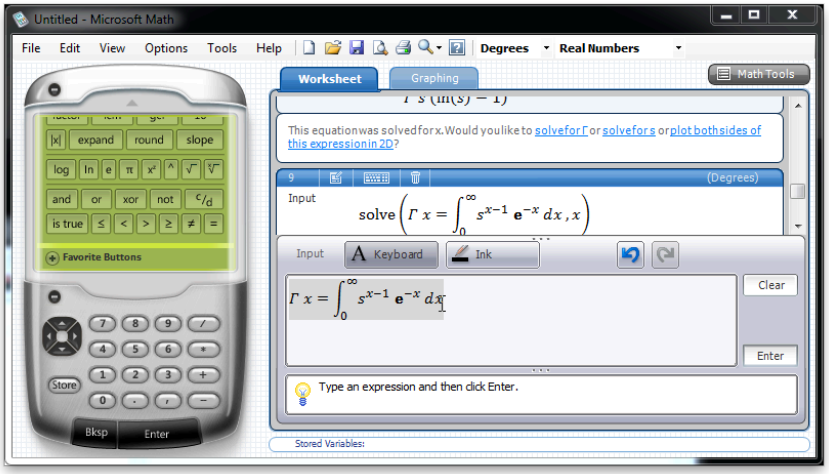


Abbildung 1.6.: Microsoft Math 3.0 Tastatureingabe

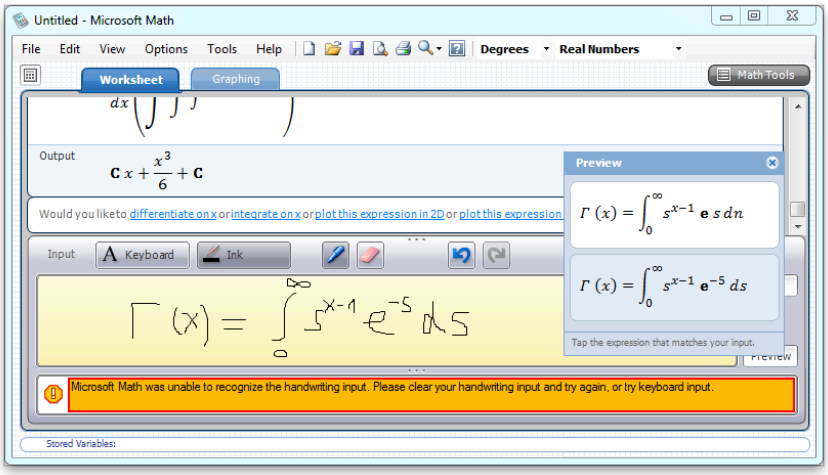


Abbildung 1.7.: Microsoft Math 3.0 Handschrifterkennung

1.2.6. MathJournal

XThink [53] vertreibt eine weitere kommerzielle Software namens MathJournal. Sie ist an Mathematiker und Ingenieure gerichtet und hat das Ziel das Lösen von mathematischen Problemen auf einem Tabletcomputer zu erleichtern. Es gibt leider keine Evaluations-Version, daher konnte ich mich nicht von der Leistungsfähigkeit überzeugen. Tapia u. Rojas [42] schreiben aber, dass MathJournal die Microsoft API zur Erkennung von Symbolen verwendet. Die Leistungsfähigkeit wird also ähnlich sein wie die von Microsoft Math. Der Knackpunkt wird also wieder die Qualität der Werkzeuge zur Fehlerkorrektur sein. Die Abbildung 1.8 gibt einen Eindruck von MathJournal.

1.2.7. Weitere

Die aktuelle Literatur beschreibt weitere Systeme, die Wege erforschen, interaktive Mathematik auf dem Computer durch natürliche handschriftliche Eingabe zu erleichtern. Dabei ist MathBrush [25] zu erwähnen, das ein experimentelles Projekt ist um die handschriftliche Eingabe für Formeln in ein Computer Algebra System (CAS) zu ermöglichen und dann zu manipulieren. Vuong u. a. [47] stellen ein webbasiertes Handschrift-Mathematik-System vor, das eine flexible mobile Umgebung zum Lösen mathematischer Probleme bieten soll. Als Backend wird Maple [3] verwendet. Sie gehen also nach der Erkennung weiter als Programme wie Microsoft Math und MathJournal, die eher als erweiterter Taschenrechner mit Handschrifteingabe zu sehen sind.

Für beide Projekte bleiben die Probleme der eigentlichen Eingabe natürlich dieselben.

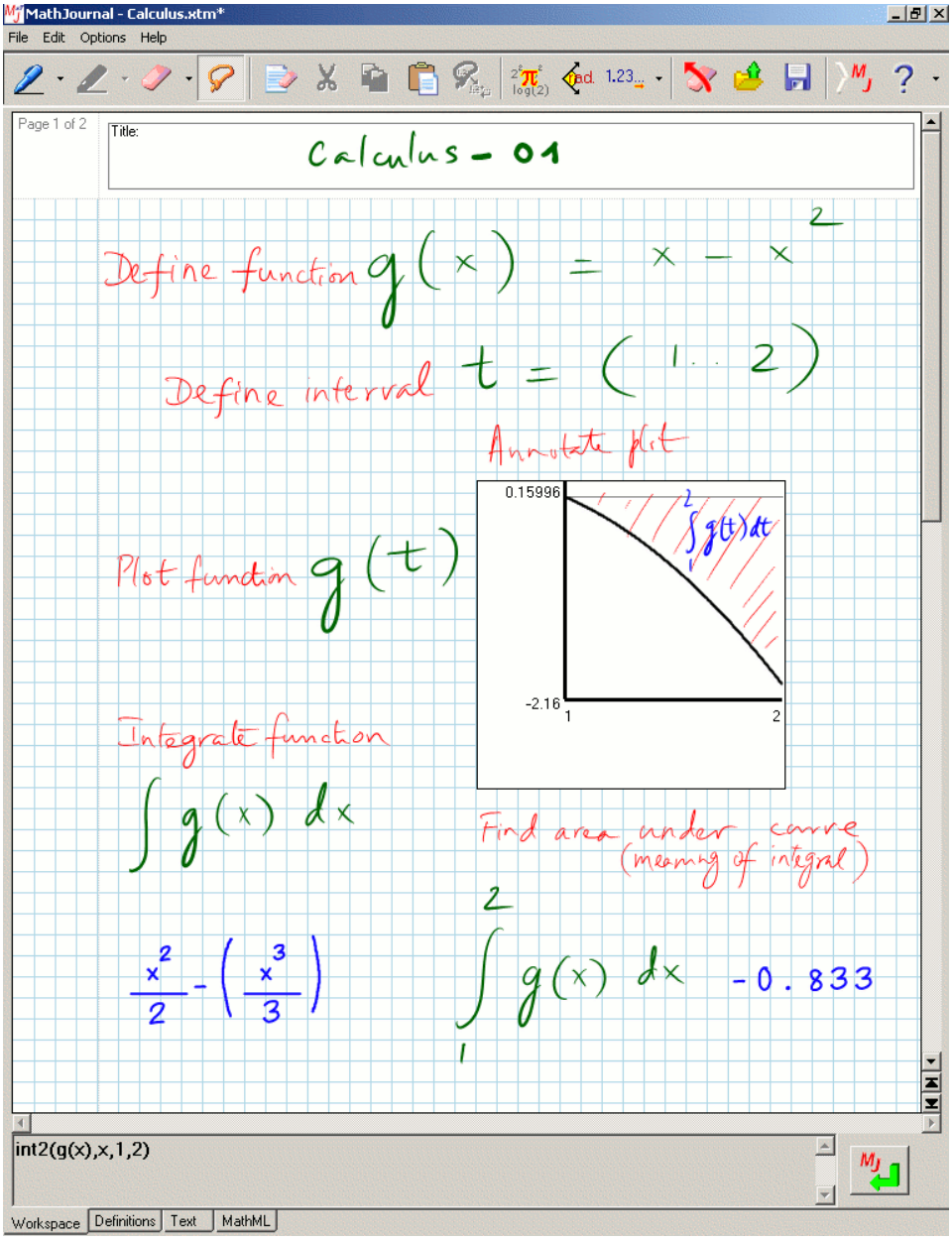


Abbildung 1.8.: MathJournal

1.3. Was ist so schwierig an Formelerkennung?

Die vorgenannten Werkzeuge schneiden deswegen nicht gut ab, weil die Erkennung mathematischer Formeln ein wirklich schwieriges Problem ist. Es teilt sich in drei Teile auf. Der erste Teil ist die Segmentierung der mathematischen Formel, bei der einzelne Symbole isoliert werden müssen. Als Zweites müssen die einzelnen Symbole richtig erkannt werden. Schließlich müssen die Symbole über ihre räumliche Position in eine logische Beziehung zueinander gebracht werden. Die in diesen Schritten gewonnenen Erkenntnisse können natürlich die Entscheidung in den jeweils anderen beeinflussen, indem man die Semantik einer Interpretation in die Erkennung mit einfließen lässt.

Das Problem ist also komplex. Entsprechend treten häufig Erkennungsfehler auf und diese zu korrigieren ist mit allen getesteten Programmen nicht ideal gelöst. Im Gegenteil ist der Korrekturvorgang eher langwierig und frustrierend. Tapia u. Rojas [42] geben eine Übersicht, die die Problematik genauer beleuchtet.

1.4. Suche nach Alternativen

Die optimale Eingabemethode ist also (noch) nicht perfekt umgesetzt. Um ganze Formeln zuverlässig zu erkennen wurden noch nicht die optimalen Erkennungsalgorithmen und das optimale Benutzerinterface zum Umgang mit Fehlern gefunden. Eine Alternative ist, dem Computer nur einen Teil der Erkennung zu übertragen. Ein Beispiel hierfür ist JEquation [2]. In diesem Programm wird dem Benutzer vorgegeben, wo er zu malen hat, damit die Struktur der Formel richtig erkannt wird. Es bleibt also die Aufgabe, die einzelnen Symbole zu erkennen. Abbildung 1.9 zeigt dieses Benutzerinterface.

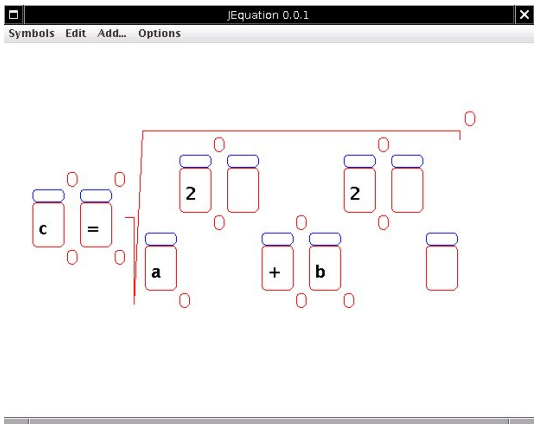


Abbildung 1.9.: JEquation

Es gibt natürlich auch Ansätze, die ohne jede Form von Mustererkennung auskommen.

Hier ist Lyx ein Beispiel. Lyx ist ein What You See Is What You Mean (WYSIWYM)-Editor. Er funktioniert also ähnlich wie ein WYSIWYG-Editor wie Word. Dabei arbeitet man nicht direkt mit LaTeX-Befehlen, sondern hat einen grafischen Editor, in dem man jedoch die Textabschnitte semantisch auszeichnet, statt den Schriftstil manuell vorzugeben. Für Mathematische Formeln enthält Lyx einen Formeleditor, der ähnlich dem funktioniert, was aus Office-Anwendungen bekannt ist. Um Symbole einzufügen hat man einerseits die Möglichkeit LaTeX-Befehle (mit Auto-Vervollständigung) direkt einzugeben, oder das gesuchte Symbol aus mehreren Symboltabellen auszuwählen und per Klick einzufügen. Die erste der beiden Möglichkeiten setzt natürlich wieder voraus, dass der Autor den Namen des Befehls kennt. Die zweite Methode hat den Nachteil, dass die Symboltabellen schnell unübersichtlich werden, wenn sie zu umfangreich sind. Es lässt sich also nur ein kleiner Teil der verfügbaren Symbole unterbringen, ohne den Nutzen zu schmälern.

1.5. Ein pragmatischer Ansatz

Viele Benutzer fühlen sich aber durchaus wohl damit ihre Dokumente direkt im Quelltext zu bearbeiten. Texteditoren wie Vim, Emacs etc. erfreuen sich großer Beliebtheit zum Erstellen von LaTeX-Dokumenten [4, 5]. Ich selbst ziehe den Texteditor TextMate trotz umfangreicher Recherchen in diesem Bereich jedem spezialisierten LaTeX-Editor vor, da ich mit diesem viel effizienter arbeiten kann. Die Frage ist also, wie ein pragmatisches Werkzeug aussieht, das einem typischen Anwender die Arbeit an seinem Dokument insbesondere die Eingabe von mathematischen Formeln erleichtert, ohne ihm eine Arbeitsweise vorzuschreiben.

Ein Problem, das jeder Anwender egal ob Einsteiger oder Fortgeschrittener einmal hat, ist, dass er den Befehl für ein Symbol nicht weiß. Entweder er hat ihn noch nie gebraucht, oder er ist ihm entfallen. Das Symbol ist auch nicht in den Symboltabellen in seinem LaTeX-Editor (falls er nicht sowieso einen Texteditor verwendet) vorhanden. Also muss er ein Buch oder die „Comprehensive LaTeX Symbol List“ [31] wälzen, um das gewünschte Symbol zu finden. Dies ist aufgrund der großen Anzahl an Symbolen sehr zeitaufwändig. [31] listet immerhin 4947 unterschiedliche Symbole auf. Genau hier könnte ein pragmatisches Werkzeug ansetzen und auf einem Mittelweg zwischen der reinen Texteingabe und vollständiger Formelerkennung eine Lösung für dieses Problem anbieten.

Das Werkzeug müsste also eine Symbolsuche bieten, durch die sehr schnell der Befehl zum gesuchten Symbol gefunden werden kann. Der Benutzer kann sein Dokument im Editor seiner Wahl bearbeiten und im Bedarfsfall das Suchwerkzeug aufrufen, um schnell wieder an die eigentliche Arbeit, das Verfassen des Dokuments, zu gehen. Dieses Werkzeug soll im Folgenden vorgestellt werden.

2. Detexify – LaTeX-Symbolsuche als Webanwendung

In diesem Kapitel beschreibe ich die Architektur, Funktionalität und Implementierung von Detexify, der Anwendung zur LaTeX-Symbolsuche, die ich im Rahmen dieser Arbeit entwickelt habe. Detexify kann mit einem modernen Browser[1] unter der Adresse `http://detexify.kirelabs.org` aufgerufen werden. In Anhang D findet sich ein kurzes Benutzerhandbuch.

2.1. Architektur

Bei jeder Anwendung muss man sich, bevor sie geschrieben wird, entscheiden, wie die Anwendung zur Verfügung gestellt werden soll. Daraus resultieren weitere Entscheidungen, wie z.B. die Wahl der Programmiersprache und der Datenformate.

Detexify wurde als Webanwendung implementiert. Dies hat die folgenden Vorteile:

- **Plattformunabhängigkeit:** Heutzutage ist ein Webbrowser auf jedem Computer verfügbar. Um Detexify verwenden zu können wird lediglich ein moderner Webbrowser benötigt.
- **Zentrale Wartung:** Fehlerbehebungen und Verbesserungen sind zentral anwendbar und stehen jedem Benutzer beim nächsten Gebrauch der Anwendung sofort zur Verfügung.
- **Zentrale Trainingsdaten:** Die Trainingsdaten sind zentral in einer Datenbank gespeichert. Das Training des Klassifizierers wird von den Nutzern selbst durchgeführt (siehe auch 2.4).

Dabei besteht Detexify aus zwei lose gekoppelten Komponenten, und zwar der Webanwendung und dem Klassifizierungsserver (Server). Lose Kopplung heißt dabei, dass lediglich ein Interface definiert ist, wie die Webanwendung mit dem Server zu kommunizieren hat [46]. Das Interface ist als Representational State Transfer (REST)-Interface [12] spezifiziert. Die beiden Komponenten kommunizieren also über das Hypertext Transfer Protocol (HTTP) [11]. Abbildung 2.1 zeigt die Architektur schematisch.

[1] Aktuelle Versionen von Firefox, Chrome, Safari und Opera

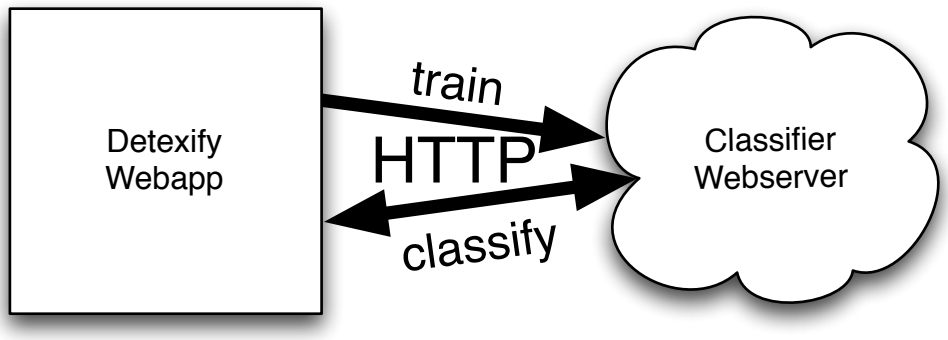


Abbildung 2.1.: Detexify Architektur

2.2. Webanwendung

Die Webanwendung stellt die Benutzeroberfläche von Detexify dar. Sie wird über einen Browser bedient. Es gibt zwei Ansichten (jeweils durch statische HTML-Seiten und JavaScript realisiert). Die erste ermöglicht die Symbolsuche und die zweite bietet eine Symboltabelle aller in Detexify registrierten LaTeX-Symbole. In dieser können die Symbole auch trainiert werden.

2.2.1. Symbolsuche

Die Symbolsuche ist das Zentrum der Anwendung. Sie ist das Werkzeug, das dem LaTeX-Autor das Leben vereinfachen soll. Abbildung 2.2 zeigt einen Screenshot eines Browsers mit geöffneter Symbolsuche.

Auf der Seite der Webanwendung ist dafür eine Zeichenfläche implementiert. Auf dieser können mit der Maus (oder einem Grafiktablett) Striche gemalt werden, und nach jedem beendeten Strich wird per AJAX [49] eine Erkennungsanfrage an den Server gesendet. Die Symbole werden dann anhand der vom Server ermittelten Rangfolge aufgelistet (wie in Abbildung 2.2 zu sehen). Die Kommunikation erfolgt wie in 2.3 beschrieben über das Datenformat JavaScript Object Notation (JSON) [9]. Die Interaktionen in der Benutzeroberfläche (Zeichnen usw.) sind mithilfe von JavaScript gelöst. Die Zeichenfläche basiert auf der Scalable Vector Graphics (SVG)-Technologie [48].

2.2.2. Symboltabelle

Einerseits bietet die Symboltabelle einen Überblick über die in Detexify registrierten und damit über die Symbolsuche auffindbaren Symbole. Andererseits bietet sie durch ihre Sortierungs- und Filtermöglichkeiten von sich aus gute Möglichkeiten zur Symbolsuche,

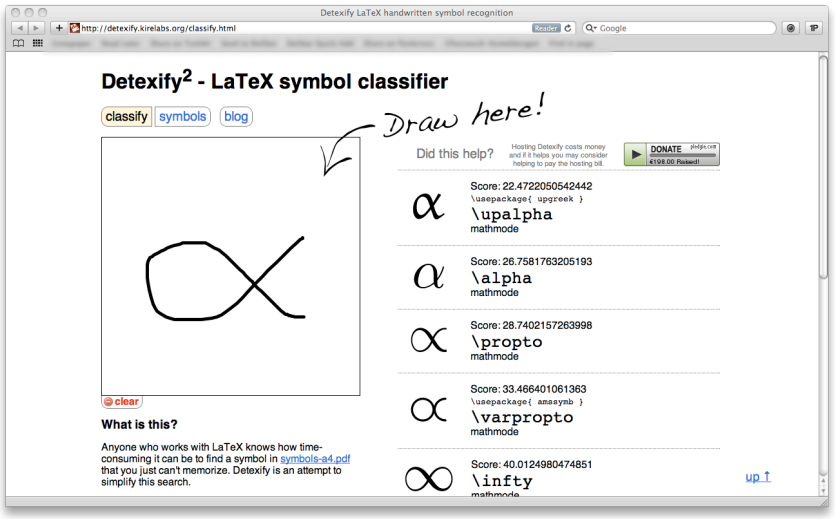


Abbildung 2.2.: Symbolsuche

sollte die eigentliche Symbolsuche versagen. Außerdem können in der Symboltabelle gezielt einzelne Symbole trainiert werden. Die Funktionalität der Symboltabelle ist im Detail in Anhang D.2 beschrieben. Die verwendeten Technologien sind identisch mit den bei der Symbolsuche verwendeten. Abbildung 2.3 zeigt einen Screenshot eines Browsers mit geöffneter Symboltabelle.

2.3. Server

Der Server übernimmt die Aufgabe der Mustererkennung. An ihn können im Wesentlichen zwei verschiedene Anfragen gestellt werden. Die erste davon ist das Training einer Symbolklasse. Die zweite ist die Klassifizierung von unbekannten Daten. Es gibt noch eine dritte Anfrage, die aber lediglich zur Identifizierung und Statusermittlung der Serverimplementierung dient.

Da der Server die eigentliche Arbeit verrichtet läuft er auf einem leistungsstarken Rechner. Wie die eigentliche Symbolerkennung aussieht, ist in Kapitel 3 beschrieben, während 2.3.1 das REST-Interface spezifiziert.

2.3.1. REST-Interface

Training

Um eine Symbolklasse zu trainieren müssen dem Server zwei Informationen übergeben werden und zwar den Bezeichner der Klasse und die Trainingsdaten. Der Bezeichner ist

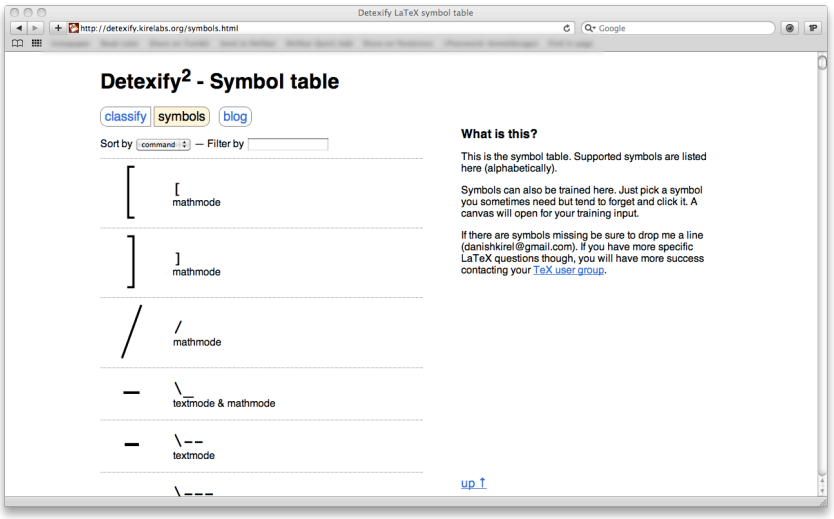


Abbildung 2.3.: Symboltabelle

dabei Teil des Uniform Resource Locator (URL), und die Trainingsdaten werden als JSON-String [9] im HTTP-Request-Body übertragen. Wie die Daten genau kodiert sind, ist in 3.3 beschrieben.

Listing 2.1: Anfrage

```
POST /train/{id}
application/json
[[{"x":12.3, "y":4.56, "t":7890},...],...]
```

Listing 2.2: Antwort

```
200 OK
```

Listing 2.3: Antwort im Fehlerfall

```
422 Unprocessable Entity
application/json
{ "error" : "error message" }
```

Erkennung

Zur Erkennung werden die unbekannten Daten als JSON-String an den Server übertragen. Dies geschieht wie beim Training im HTTP-Request-Body. Als Antwort erhält die Webanwendung eine JSON-kodierte Liste von Klassenbezeichnern und einer zugehörigen Wertung. Je kleiner die Wertung, desto wahrscheinlicher die zugehörige Klasse.

Listing 2.4: Anfrage

```
POST /classify
application/json
[[{"x":12.3, "y":4.56, "t":7890},...],...]
```

Listing 2.5: Antwort

```
200 OK
application/json
[{ "id" : "key", "score": 1.234}, ...]
```

Listing 2.6: Antwort im Fehlerfall

```
422 Unprocessable Entity
application/json
{ "error" : "error message" }
```

Serverstatus

Durch die lose Kopplung durch das hier definierte Interface lässt sich der Server leicht austauschen.[2] Um die verwendete Serverimplementierung identifizieren zu können gibt es noch eine dritte Anfrage:

Listing 2.7: Anfrage

```
GET /
```

Listing 2.8: Antwort

```
200 OK
application/json
{
  "server" : "server string to identify implementation",
  "version" : "version string e.g. 1.0"
}
```

[2] Tatsächlich existieren Serverimplementierungen in den Programmiersprachen Haskell, Ruby und Clojure.

2.4. Crowdsourcing des Trainings

Wie in 2.2.2 erwähnt, bietet Detexify den Nutzern eine Oberfläche an, um das Training durchzuführen. Tatsächlich habe ich nie ein initiales Training vorgenommen. Der Plan war von Anfang an, das Training vollständig den Nutzern zu überlassen. So eine Vorgehensweise nennt man Schwarmauslagerung (Crowdsourcing) [19]. Die Überlegung war dabei die Folgende: Wenn die Nutzer von guten Trainingsdaten durch bessere Erkennungsraten profitieren, sind sie auch bereit ein wenig Arbeit zu investieren, damit einige von ihnen ausgewählte Symbole leicht gefunden werden. Durch die Streuung der Auswahl der Nutzer wird eine große Menge an Symbolen in kurzer Zeit auf einen guten Trainingsstand gebracht. Ein Unternehmen, das sicherlich heutzutage jedem bekannt ist und genau eine solche Strategie verfolgt, ist das Online-Lexikon Wikipedia. Wikipedia wird vollständig von seinen Nutzern gepflegt und ist trotzdem, oder eher genau deshalb „heute wahrscheinlich die beste Enzyklopädie der Welt: Größer, aktueller und in vielen Fällen umfangreicher als die *Encyclopædia Britannica*" [6, S.83]

Der Plan Crowdsourcing für das Training zu verwenden war sehr erfolgreich. Als Detexify am 11. Juli 2009 online ging, war die Idee völlig neu. Eine LaTeX-Symbolsuche in dieser Form hatte zu diesem Zeitpunkt noch niemand gesehen, daher verbreiteten sich Links zu Detexify in Windeseile über soziale Netze wie Twitter, Facebook und Delicious, aber auch über Newsportale wie Reddit und Hacker News, und am 14. Juli 2009 erreichte Detexify seinen Spitzenwert von über 10.000 Besuchern an einem Tag. Erst gegen Ende Juli 2009 normalisierten[3] sich die Besucherzahlen. Bis zu diesem Zeitpunkt hatten die Nutzer bereits mehrere tausend Trainingsbeispiele gespendet. Zur Zeit liegt die Anzahl der Trainingsbeispiele bei fast 150.000 für 977 Symbole.

Crowdsourcing hat natürlich auch seine Nachteile. Für die Benchmarks in Kapitel 4 habe ich einen Spiegel der Datenbank erstellt, der zu diesem Zeitpunkt 147.881 Trainingsbeispiele enthielt. Bei der manuellen Inspektion der Daten stellte sich schnell heraus, dass sie für manche Symbole eine katastrophale Qualität hatten. Ein besonders krasses Beispiel ist das Symbol `\male` (♂), bei dem ca. 200 der 533 Trainingsbeispiele nicht Zeichnungen dieses Symbols sondern von männlichen Geschlechtsteilen waren. Insgesamt habe ich bei der manuellen Inspektion aller 147.881 Trainingsbeispiele 14.419 Zeichnungen, die völlig aus dem Rahmen fielen, aussortiert.[4] Die Benchmarks in Kapitel 4 basieren auf diesen grob bereinigten Daten.

[3] Angang B gibt einen Einblick in die aktuelle Benutzerzahlen.
[4] Die Trainingsbeispiele enthielten aber auch einige künstlerische Perlen, die in Anhang E verewigt sind.

3. Erkennung handgeschriebener Symbole

Die Erkennung handgeschriebener Symbole ist das Herzstück von Detexify. Hier wird einer unbekannten Eingabe von Daten eine Rangfolge von LaTeX-Symbolen zugeordnet. Wie bei jeder Mustererkennungsaufgabe steht dabei auch hier der Erkennungsalgorithmus im Zentrum. In 3.5 stelle ich die möglichen Algorithmen vor und analysiere sie auf ihre Tauglichkeit für Detexify.

Je nach verwendetem Verfahren gehen der Klassifizierung eine Vorverarbeitung (3.4 und 3.6) und Merkmalsextraktion (3.7) voraus.

Das Kapitel beginnt nun mit einer Einführung in die Terminologie der Handschrifterkennung und die Problematik der mathematischen Symbolerkennung.

3.1. Terminologie

Wie in jedem Forschungsgebiet hat sich auch bei der Handschrifterkennung eine Nomenklatur entwickelt, die die verschiedenen Aspekte des Themas beschreibt.

Man unterscheidet zwischen *Online*- und *Offline*-Systemen [45]. Bei Online-Systemen wird die Erkennung durchgeführt während der Nutzer schreibt. Die Striche werden dabei vom Eingabegerät (Grafiktablett oder Computermaus) als Funktion $t \mapsto \alpha_t$ an das System übertragen, wobei α_t den Zustand der Stiftspitze zum Zeitpunkt t kodiert. Es stehen also für die Erkennung alle dynamischen Eigenschaften des Geschriebenen zur Verfügung, wie die Anzahl, Reihenfolge und Richtung der einzelnen Pinselstriche. Im Gegensatz dazu verarbeiten Offline-Systeme Scans von Geschriebenem, arbeiten also zu einem Zeitpunkt, wenn der Schreibvorgang längst beendet ist. Sie können also außer Pixeln keine weiteren Informationen verwenden. Detexify ist ein Online-System.

Der Zustand der Stiftspitze besteht in Online-Systemen in der Regel aus den Koordinaten (x, y), der Information, ob die Spitze das Tablett berührt, oft bezeichnet mit *pen-up* bzw. *pen-down* und in manchen Fällen auch aus der Neigung und dem Azimut.

Aus den Daten werden dann häufig Merkmale abgeleitet, sog. *Features*. Man unterscheidet zwischen *globalen* und *lokalen* Features [42]. Lokale Features sind solche, die von einzelnen Punkten auf einem Strich und dessen benachbarten Punkten abgeleitet werden. Globale Features werden hingegen von der Menge der Striche als Ganzes abgeleitet.

3.2. Herausforderungen der Erkennung von LaTeX-Symbolen

Die Erkennung von handgemalten LaTeX-Symbolen ist allein durch die enorme Anzahl der Symbole eine große Herausforderung [23]. Die Symbole können aus sehr unterschiedlichen Alphabeten kommen. Es kommen lateinische, griechische, mathematische und weitere Symbole vor. Im Gegensatz zu asiatischen Sprachen, die ebenfalls sehr große Alphabete aufweisen[1], ist jedoch die Anzahl und Reihenfolge der Striche nicht vorgegeben, was die Erkennung zusätzlich erschwert [50]. Außerdem gibt es sehr viele sehr ähnliche Symbole wie $\to, \mapsto, \rightsquigarrow, \rightharpoonup, \hookrightarrow, \rightarrowtail$. Dazu kommen noch die Probleme, vor denen jede Handschrifterkennung steht, wie unterschiedliche Schreibstile sowohl von unterschiedlichen Schreibern als auch natürliche Variationen in der Schreibweise eines Einzelnen. Abbildung 3.1 zeigt als Beispiel drei Trainingsbeispiele des Symbols π.

In den Trainingsbeispielen in Detexify ist mit einer noch höheren Variation zu rechnen, da in den meisten Fällen eine herkömmliche Computermaus zum Zeichnen der Symbole Verwendung findet statt ein Grafiktablett, das eine gewohnte Stiftführung ermöglicht.

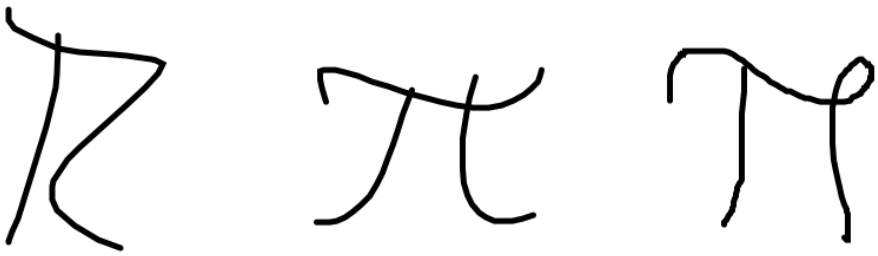


Abbildung 3.1.: Drei Trainingsbeispiele für das Symbol π

3.3. Eingabe

Die in 2.2.1 beschriebene Zeichenfläche erlaubt sowohl die Eingabe über ein Grafiktablett als auch per Maus, wobei letzteres jeder Computeranwender zu Hause hat. Die Eingabe erfolgt in der Webanwendung, und die Daten, die an Server gesendet werden und damit den Erkennungsalgorithmus erreichen, sind durch das REST-Interface (siehe 2.3.1) spezifiziert. Die Daten bestehen aus einer Liste von Strichen. Dabei besteht jeder Strich aus einzelnen Punkten, die die Bewegung der Stiftspitze beschreiben. Ein Punkt beinhaltet seine Koordinaten x und y sowie einen Zeitstempel t. Im JSON-Format sieht das wie folgt aus:

[1] In Japan werden heute 6000-7000 Buchstaben verwendet. In China liegt die Anzahl der im täglichen Leben verwendeten Buchstaben bei etwa 5000 [20]

```
[[{"x":125.5, "y":219.133331298828, "t":1269620535913}, ...], [...]]
```

3.4. Vorverarbeitung und Merkmalsextraktion

Bevor die Daten an die Erkennungsalgorithmen ausgeliefert werden, empfiehlt es sich eine Vorverarbeitung (Preprocessing) durchzuführen. Dies ist eine wirksame Methode um Rauschen, das z.B. durch Ungenauigkeiten der Eingabegeräte oder Unachtsamkeit des Schreibers entstehen kann, zu relativieren. Eine Vorverarbeitung kann aber auch überflüssige Informationen entfernen, die vom Erkennungsalgorithmus nicht verwendet werden. Zudem kann durch Normalisierung der Daten ungewollte Variation reduziert werden. Es gibt kaum ein System zu Handschrifterkennung, das keine Vorverarbeitung durchführt [20, 32, 45].

Merkmale sind (häufig numerische) Werte, die aus den Mustern extrahiert werden. Sie werden dann in der Regel in einem Vektor, dem Merkmalsvektor, zusammengefasst. Der Vektorraum, in dem diese Vektoren liegen, heißt dann der Merkmalsraum.

Da die Art der Vorverarbeitung sowie die Merkmalsextraktion vom Erkennungsalgorithmus abhängt, werden die in Detexify ergriffenen Maßnahmen erst in 3.6 und 3.7 nach der Analyse der Erkennungsalgorithmen erläutert.

3.5. Klassifizierung

Zur Erkennung einzelner handgeschriebener Symbole wurden bereits unterschiedlichste Klassifikationsverfahren verwendet [32]. Um ein leistungsstarkes Backend für Detexify zu entwickeln muss also ein Verfahren ausgewählt und gegebenenfalls optimiert werden, das den spezifischen Anforderungen der Anwendung und der Architektur gerecht wird.

Die Anforderungen sind die folgenden:

Adaptionsfähigkeit Es sollte jederzeit ein Training zusätzlicher Symbole möglich sein.

Skalierbarkeit Die Erkennungsraten sollten auch bei einer großen Anzahl von Klassen gut sein.

Laufzeitverhalten Das Laufzeitverhalten sollte eine Erkennung in Echtzeit ermöglichen.

Interaktivität Es sollten die besten N Symbole zur Auswahl angezeigt werden.

In diesem Abschnitt untersuche ich verschiedene Verfahren auf ihre Tauglichkeit für Detexify. Ein besonderes Interesse gilt in meinem Fall den Verfahren, die für Symbolerkennung insbesondere mathematischer Symbole bereits erfolgreich eingesetzt wurden. Dabei handelt es sich um Support Vector Machines (SVM) (3.5.3), Hidden Markov Model (HMM) (3.5.4) und template matching (3.5.5) mit Dynamic Time Warping (DTW) (3.5.6). Es gibt auch einige wenige strukturelle Methoden (3.5.1).

Es wird sich bei der Untersuchung zeigen, dass *template matching* zusammen mit *DTW* am besten geeignet ist.

3.5.1. Strukturelle Methoden

Strukturelle Methoden basieren auf der Annahme, dass Handschrift aus elementaren Formen, auch Allographen genannt, besteht. Die Darstellung einer Klasse erfolgt dann als strukturelle Repräsentation (z.B. als Baum oder Graph) bestehend aus diesen Allographen. Die Vorgehensweise lässt sich am besten an einem Beispiel illustrieren. Fitzgerald u. a. [14] stellen mathematische Symbole als eine Kombination von Merkmalen der Typen *Linie*, *C-Form* oder *O-Form* dar. Diese Merkmale werden mithilfe von Fuzzylogik aus den Strichen extrahiert, und zur Zuordnung des richtigen Labels kommt wieder Fuzzylogik zum Einsatz. Dabei haben sie für jede verwendete Symbolklasse, eine eigene Fuzzy-Regel erstellt. Im zitierten Artikel waren das Regeln für die Zahlen 1-9 und kleingeschriebene lateinische Buchstaben. Es ist offensichtlich, dass ein solches Vorgehen für Detexify mit nahezu 1000 Symbolklassen nicht in Frage kommen kann. Zum Problem für jede Klasse ein Modell zu definieren kommt hinzu, dass das System nicht *adaptionsfähig* im in 3.5 definierten Sinne ist, da jedes neue Symbol ein neues Modell benötigt, das von Hand hinzugefügt werden müsste. Ohnehin ist es schwierig bei einer so großen Anzahl von Symbolklassen eine gemeinsame Struktur zu finden, die einen strukturellen Ansatz praktikabel macht. Aus den genannten Gründen habe ich strukturelle Methoden für Detexify verworfen.

3.5.2. Künstliche Neuronale Netze

Neuronale Netze haben bekanntlich Schwierigkeiten, wenn die Anzahl der Klassen groß ist [20]. Dementsprechend ist die Anzahl der in Übersichtsartikeln zur Erkennung von mathematischen Formeln wie [7] und [42] zitierten Arbeiten, die neuronale Netze verwenden, äußerst gering. Ich habe zudem keinen nach 1996 veröffentlichten Artikel gefunden, der neuronale Netze für diese Aufgabe verwendet. Neuronale Netze habe ich deshalb für Detexify nicht weiter in Betracht gezogen.

3.5.3. Support Vector Machines

SVMs [37] haben sich zu einem beliebten Klassifikationsverfahren für bestimmte Problemklassen entwickelt. Bei dem Verfahren betrachtet man eine Menge von (Trainings-) Objekten in einem Verktorraum (dem Merkmalsraum). Jedes Objekt gehört einer Klasse an, und nun legt man eine Hyperebene so durch den Raum, dass die Klassen getrennt und der Abstand zu beiden Seiten der Hyperebene maximal wird. Eine SVM ist dabei als Funktion realisiert, die die Lage eines Eingabevektors zur Hyperebene zurückgibt. Das Ergebnis ist besser, wenn der Eingabevektor weit auf der richtigen Seite liegt, also der Abstand zur Ebene größer ist. Das Verfahren ist natürlich nur sinnvoll, wenn sich die Klassen linear trennen lassen. Ist dies nicht der Fall, bedient man sich des Kernel-Tricks [37, S.15] um den Merkmalsraum auf einen höherdimensionalen Verktorraum abzubilden, in welchem die Wahrscheinlichkeit, dass die Klassen linear trennbar sind, höher ist.

SVMs sind binäre Klassifizierer. Um sie also bei $(N > 2)$ Klassen verwenden zu können, gibt es unterschiedliche Strategien. Zum einen gibt es die Winner-takes-all (WTA)-Strategie, bei der N binäre SVMs konstruiert werden, wobei die i-te SVM ρ_i die Klasse C_i und $C_i^{\complement}$ (also alle anderen Klassen) unterscheidet. Einem unbekannten Objekt x wird dann die Klasse zugeordnet, deren SVM ρ_i das beste Ergebnis zurückliefert. Zum anderen gibt es die Max-wins voting (MWV)-Strategie, bei der für jedes Paar von Klassen C_i und C_j eine SVM ρ_{ij} mit den Objekten aus C_i gegen die Objekte aus C_j trainiert wird. Für ein unbekanntes Objekt x werden nun alle $N(N-1)/2$ Klassifizierer nach ihrer Stimme gefragt, und die Klasse mit den meisten Stimmen gewinnt. Weitere Kombinationsstrategien finden sich in [10] und [33].

Tapia u. Rojas [40, 41] verwenden directed acyclic graph-SVMs (DAGSVMs) [33] zur Symbolerkennung in einem System, das u.a. mathematische Formeln auf einer elektronischen Tafel erkennt. Golubitsky u. Watt [15], Keshari u. Watt [22], Golubitsky u. Watt [17] stellen Mathematische Symbole als Merkmalsvektor bestehend aus den Koeffizienten einer Funktionalapproximation der Striche dar und trainieren damit SVMs, die sie mit MWV kombinieren. Die genannten Autoren berichten auch von guten Ergebnissen — jedoch hätten SVMs für Detexify einen entscheidenden Nachteil:

Das Problem besteht darin, dass SVMs schlecht nachtrainiert werden können. Um nachträglich ein neues Trainingsmuster hinzuzufügen, müssen bei N Klassen beim deutlich besseren MWV-Verfahren [10] $(N-1)$ SVMs angepasst (das heißt die Hyperebenen neu ausgerechnet) werden. Um eine neue Klasse hinzuzufügen müssen ebenfalls N neue SVMs eingefügt werden. Ein inkrementelles Training ist also bei SVMs gar nicht denkbar. SVMs sind also ebenfalls nicht *adaptionsfähig* und kommen daher nicht in Frage.

3.5.4. Hidden Markov Models

HMM sind ein statistisches Mustererkennungsverfahren, das schon seit langem in der Spracherkennung eingesetzt wird [35]. Das liegt daran, dass es besonders geeignet ist zeitliche Muster zu modellieren und außerdem Segmentierung und Klassifizierung integriert [24]. Der Merkmalsvektor darf dabei von Muster zu Muster in der Länge variieren. Dabei beschreibt ein HMM einen diskreten Markovprozess $q = (q_t)_{t=0,1,\ldots}$, dessen Zustandsfolge aber nicht sichtbar (also verborgen, daher der Name) ist. Stattdessen sieht der Beobachter eine Symbolfolge $\mathbf{O} = O_1 \ldots O_T$ mit $O_t \in V$ aus einem endlichen Alphabet $V = \{v_i\}_{i=1\ldots K}$, wobei zu jedem Zeitpunkt t jeweils O_t nur vom verborgenen Zustand q_t der diskreten Markovkette abhängt. Abb. 3.2 illustriert das Modell. Eine ausführliche Einführung zu HMMs inklusive Erklärung von diskreten Markovprozessen findet sich bei Rabiner [35].

Man verwendet ein HMM dann folgendermaßen: Jede Klasse C_i bekommt ein HMM λ_i, deren Parameter anhand einer Stichprobe trainiert werden. Einem unbekannten Objekt, das als Folge $\mathbf{O}$ von Merkmalsvektoren beobachtet wird, wird dann die Klasse zugeordnet, für die die Wahrscheinlichkeit $P_{\lambda_i}(\mathbf{O})$, dass die Folge von der zugehörigen HMM erzeugt

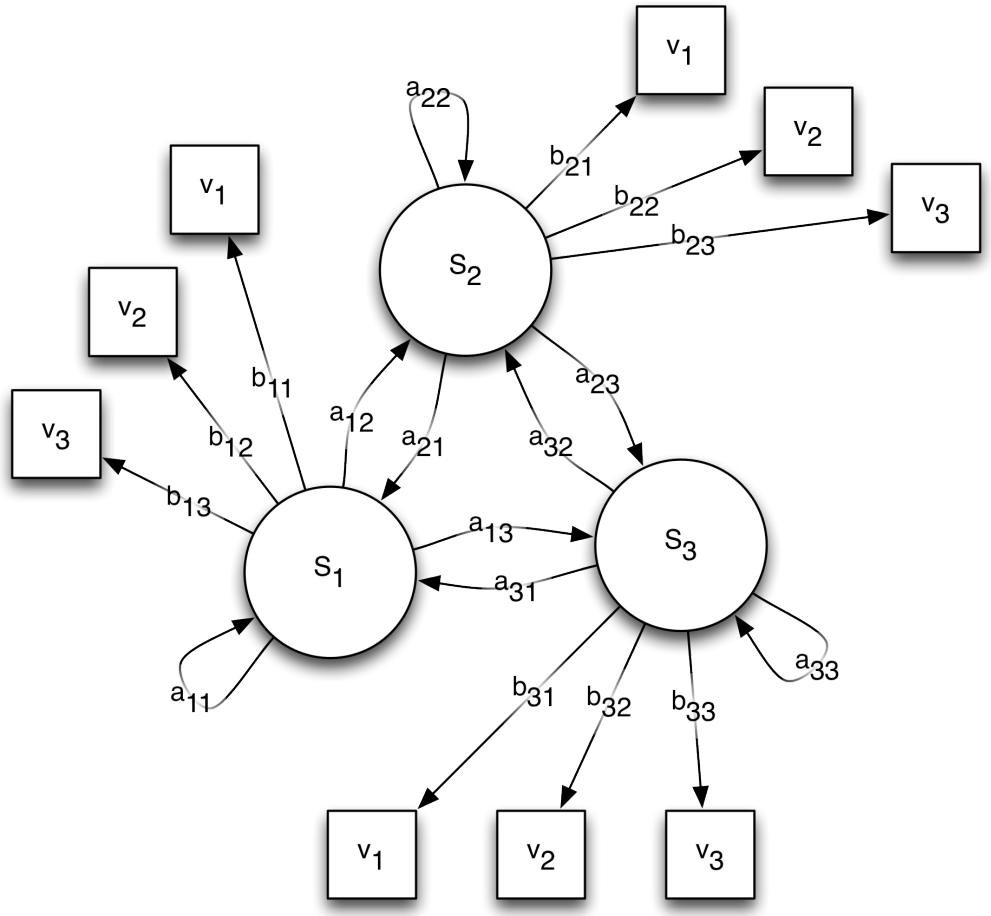


Abbildung 3.2.: HMM - mit Zustandsmenge $\{S_i\}$, Übergangswahrscheinlichkeiten $\{a_{ij}\}$, Ausgabealphabet $\{v_i\}$ und Ausgabewahrscheinlichkeiten $\{b_{ij}\}$

wurde, maximiert wird.

Xie [52] beschreibt ein System zur Erkennung mathematischer Symbole, das auf HMMs basiert. In derselben Arbeit wird auch ein Klassifizierer, der auf DTW basiert, und für diesen werden beeindruckende Erkennungsraten von im besten Fall 94.8 % angegeben. Obwohl sich ein Vergleich angeboten hätte werden für das HMM-basierte System jedoch keine absoluten Erkennungsraten vorgestellt. Stattdessen werden lediglich relative Veränderungen für unterschiedliche Parameterkonstellationen angegeben. Dies weckt die Vermutung, dass der zweite Ansatz nicht konkurrenzfähig ist.

Winkler [51] kombiniert mehrere HMMs, die mit unterschiedlichen Merkmalen trainiert werden, und erreicht bei einem Alphabet von 84 Symbolen im besten Fall eine Erkennungsrate von 91 %.

Von der Leistungsfähigkeit her scheinen HMMs also durchaus eine Option für Detexify zu sein. Ein Problem ergibt sich aber wieder bei der *Adaptionsfähigkeit*: Die Effektivität eines HMM hängt von seiner Modellierung ab (Anzahl der Zustände, Übergangswahrscheinlichkeiten, Ausgabewahrscheinlichkeiten), und diese müssten idealerweise für jede Symbolklasse individuell optimiert werden [13]. Das Hinzufügen einer neuen Symbolklasse umfasst also die Definition eines neuen HMM und dessen Training.

3.5.5. Nächste-Nachbarn-Klassifikation

Der k-Nächste-Nachbarn-Algorithmus [30, S.18] ist eines der einfachsten Verfahren um einem unbekannten Muster eine Klasse zuzuweisen. Dazu werden als Training Vertreter der Klassen in einem Merkmalsraum gespeichert. Einem unbekannten Muster wird dann durch ein Mehrheitsvotum der k nächsten Nachbarn eine Klasse zugeordnet. Darum wird diese Art der Klassifikation auch häufig *template matching* genannt. Eine zentrale Rolle nimmt dabei das Abstandsmaß ein, welches die Nachbarschaft bestimmt [20]. Es beeinflusst die Güte der Klassifikation und durch seine Laufzeitkomplexität den zeitlichen Aufwand des Vorgangs.

Dabei ist Nächste-Nachbarn-Klassifikation besonders bei Problemen beliebt, in denen die Anzahl an Klassen sehr hoch ist, wie beispielsweise bei japanischen Schriftzeichen [20]. Eine Analyse zur Eignung für Detexify scheint also angebracht.

Die Bedingung der *Adaptionsfähigkeit*, die ich in 3.5 gefordert hatte, die bei den bisher vorgestellten Verfahren schwierig umzusetzen war, ist bei der Nächste-Nachbarn-Klassifikation auf natürliche Weise dadurch gegeben, dass neue Trainingsmuster einfach nur gespeichert werden müssen. *Interaktivität* kann leicht realisiert werden, indem jeder Klasse ein auf dem Abstandsmaß basierender Wert zugewiesen wird. So erhält man für ein unbekanntes Muster eine Rangfolge der Klassen.

Es ist also zu klären, ob es ein Abstandsmaß gibt, das die Bedingungen *Skalierbarkeit* und *Laufzeitkomplexität* miteinander vereinen kann.

3.5.6. Dynamic Time Warping

Ein vielversprechendes Abstandsmaß wurde bereits in 3.5.4 erwähnt. Es wird *dynamic time warping* oder auch *elastic matching* genannt. Es wurde bereits erfolgreich für lateinische und chinesische Schriftzeichen eingesetzt [45]. Xie [52] erreicht damit für die Erkennung mathematischer Symbole eine Erkennungsrate von 94.8 %. Auch Golubitsky u. Watt [18] bemerken:

> Among the distance measures used for classifying handwritten mathematical symbols, the elastic matching distance is known to be one of the most accurate.

Golubitsky u. Watt [16] vergleichen DTW mit einem eigenen Abstandsmaß, das zwar deutlich schneller zu berechnen ist, aber auch 1-1,5% schlechtere Erkennungsraten liefert. Labahn u. a. [25] kombinieren DTW mit weiteren Methoden in einem System, das die handschriftliche Eingabe von Formeln in ein CAS ermöglicht. Auch Vuong u. a. [47] verwenden DTW zur Erkennung mathematischer Symbole in einer erst kürzlich veröffentlichten Arbeit zum Thema webbasiertes CAS mit handschriftlicher Eingabe. Es ist auffällig, dass gerade aktuellere Arbeiten DTW einsetzen. Obwohl das Verfahren schon seit 1982 [44] Verwendung findet, ist es immer noch konkurrenzfähig.

Aus diesen Gründen ist die Wahl für die Klassifikation in Detexify auf Nächste-Nachbarn-Klassifikation mit DTW als Abstandsmaß gefallen. Damit ist die Aufgabe also den Algorithmus so zu implementieren und zu optimieren, dass die Bedingungen *Skalierbarkeit* und *Laufzeitkomplexität* erfüllt sind.

3.6. Vorverarbeitung und Normalisierung

Mit der Festlegung auf Nächste-Nachbarn-Klassifikation mit DTW als Abstandsmaß können im Folgenden die notwendigen Maßnahmen zur Vorverarbeitung und Normalisierung der Striche erläutert werden:

Normalisierung der Größe und Position
: Die ankommenden Striche werden verschoben und skaliert, so dass sie unter Beibehaltung ihres Seitenverhältnisses[2] zentriert und maximal im Quadrat $[0,1] \times [0,1]$ liegen. Dieser Schritt ist notwendig um zwei Muster überhaupt sinnvoll mit DTW vergleichen zu können.

Glättung
: Jeder Punkt eines Strichs wird durch das arithmetische Mittel des Punktes und seiner beiden Nachbarpunkte ersetzt.

[2] Die Erhaltung des Seitenverhältnisses ist sehr wichtig. Dies musste ich feststellen, als mir in der Haskell-Implementierung etwa 4-5% Erkennungsrate fehlte, obwohl ich dachte, ich hätte alles eins zu eins aus meiner Ruby-Implementierung portiert.

Äquidistante Verteilung nach Bogenmaß

Da die Zeichenfläche im Browser in Detexify eine gewisse Abtastrate hat, die erstens Browser-abhängig und zweitens äquidistant in der Zeit ist, kann die Verteilung der Punkte auf den Strichen sehr ungleich sein. Daher werden die Punkte neu verteilt, so dass die räumliche Entfernung zwischen aufeinander folgenden Punkten gleichmäßig groß ist. Die Anfangs- und Endpunkte von Strichen werden dabei erhalten. Golubitsky u. Watt [16] geben an, dass eine solche Verteilung der Punkte bei Verwendung von DTW sogar die beste Wahl ist.

Dominante Punkte

Punkte mit geringer Krümmung tragen zur Charakteristik eines Strichs nicht viel bei, daher werden sie herausgefiltert. Es wird ein Grenzwinkel α definiert und es bleiben die Punkte erhalten, an denen sich die Richtung des Strichs um mehr als α ändert. Durch diese Reduktion in der Anzahl der Punkte wird außerdem DTW geringfügig beschleunigt.

Verkettung

Da DTW zwei Zeitreihen vergleicht, die Daten aber als Zeitreihen von Zeitreihen ankommen (Listen von Strichen) werden die Striche miteinander verkettet, so dass das Abstandsmaß direkt angewendet werden kann.[3]

Es gibt noch zwei Maßnahmen, die in der Literatur auftauchen, aber in Detexify keine Verwendung finden:

Sortierung der Striche

Wie in 3.2 erwähnt, ist die Reihenfolge der Striche bei mathematischen Symbolen einer großen Variation unterlegen. Matasakis [27] schlägt eine Sortierung der Striche vor, die dieses Problem beheben soll.

Richtungskorrektur der Striche

Dasselbe Problem tritt bei der Zeichenrichtung der Striche auf. Hier hat Matasakis [27] eine kanonische Zeichenrichtung vorgeschlagen und dreht Striche gegebenenfalls um.

Die Alternative zu den oben genannten Verfahren ist, für jede mögliche Zeichenreihenfolge und -richtung ein eigenes Modell anzulegen. Es ist klar, dass dies bei Symbolen mit vielen Strichen zu sehr vielen Modellen führen würde. Da das Training in Detexify aber ohnehin durch die Nutzer durchgeführt wird, habe ich mich bei Detexify für den folgenden Weg entschieden:

[3] Dies hört sich im ersten Moment naiv an, aber alle versuchten Alternativen haben bei der Erkennung schlechtere Ergebnisse als diese einfache Methode geliefert. Eine interessante, aber mit schlechteren Erkennungsraten verbundene Möglichkeit ist DTW als inneres Abstandsmaß für DTW zu verwenden.

Es wird keine Sortierung oder Richtungsänderung vorgenommen. Variationen in der Anzahl, Sortierung und Richtung der Striche werden bewusst in Kauf genommen. Bei der Suche nach der richtigen Klasse werden also automatisch verschiedene Variationen durchsucht, eben die, die von Benutzern eintrainiert wurden.[4]

Durch Sortierung oder Richtungsänderung kann man nämlich auch genau die Probleme schaffen, die man eigentlich unterbinden wollte: eine Sortierung oder Richtungskorrektur der Striche in zwei eigentlich ähnlichen Mustern kann nämlich durch zu nicht mehr ähnlichen Mustern führen [27]. Es würden außerdem neue Parametergeschaffen, die wieder optimiert werden müssten.

3.7. Merkmale

DTW erfordert keine Merkmalsextraktion, sondern arbeitet direkt mit den eingegebenen Strichen.

3.8. Dynamic Time Warping

Im folgenden werden der klassische DTW-Algorithmus nach Tappert [44] beschrieben und unterschiedliche Optimierungsmöglichkeiten diskutiert.

3.8.1. Der klassische DTW-Algorithmus

Seien zwei Folgen von Punkten

$$A = a_1, \ldots, a_n \tag{3.1}$$

$$B = b_1, \ldots, b_m \tag{3.2}$$

in einem Raum $S \ni a_i, b_i \; \forall \, i$ und $\delta : S \times S \rightarrow \mathbb{R}_0^+$ eine Kostenfunktion gegeben. In der Regel sind A und B Folgen in einem metrischen Raum z.B. $\mathbb{R}^n$ und δ ist eine Metrik. Betrachten wir nun die $n \times m$-Matrix

$$M = (d_{i,j})_{i=1\ldots n, j=1\ldots m} \text{ mit } d_{i,j} = \delta(a_i, b_j), \tag{3.3}$$

dann ist ein Warping-Pfad ein stetiger (im unten erläuterten Sinne) Pfad durch diese Matrix.

$$W = w_0, \ldots, w_K \qquad w_l = (i, j)_l \tag{3.4}$$

K ist dabei die Länge des Warping-Pfads W. Die Stetigkeit des Pfads unterliegt den folgenden Bedingungen:

[4] Tatsächlich haben einige (in Kapitel 4 aber nicht aufgeführte) Benchmarks gezeigt, dass eine Sortierung der Striche nicht hilft. Sowohl eine Sortierung wie Matasakis [27] sie vorschlägt, als auch nach anderen Kriterien haben die Erkennungsraten entweder nicht verändert oder sogar leicht verschlechtert.

Randbedingung $w_1 = (1,1)$ und $w_K = (n,m)$. Dadurch beginnt der Warping-Pfad in der linken unteren Ecke der Matrix und endet in der rechten oberen Ecke.

Stetigkeitbedingung Gegeben $w_k = (i,j)$ und $w_{k-1} = (i',j')$, so muss gelten $i - i' \leqslant 1$ und $j - j' \leqslant 1$. Dadurch sind nur Schritte zu benachbarten Zellen (auch diagonal) erlaubt.

Monotoniebedingung Gegeben $w_k = (i,j)$ und $w_{k-1} = (i',j')$, so muss gelten $i - i' \geqslant 0$ und $j - j' \geqslant 0$. Dadurch werden rückwärtige Schritte (nach links oder unten) verhindert.

Es gibt natürlich einige Pfade, die diese Bedingungen erfüllen. Nennen wir die Menge dieser Pfade

$$\mathcal{W} = \{W \text{ ist Warping-Pfad}\},$$

dann ist der DTW-Abstand definiert durch

$$DTW(A,B) = \min_{W \in \mathcal{W}} \frac{\sum_{i=1}^{K} d_{w_i}}{K}. \tag{3.5}$$

Dabei ist d_{w_i} das entsprechende Element der Matrix 3.3.

Diese abstrakte Definition bedeutet anschaulich, dass die Punkte in den zwei Folgen einander elastisch zugeordnet werden können, wobei es einige Beschränkungen gibt. Der erste Punkt der ersten Folge wird immer dem ersten Punkt der zweiten Folge und der letzte Punkt der ersten Folge immer dem letzten Punkt der zweiten Folge zugeordnet werden. Dazwischen müssen die Punkte zwar in der richtigen Reihenfolge bleiben und es dürfen keine übersprungen werden, aber es sind Mehrfachzuordnungen erlaubt. Gesucht wird die Zuordnung, die den Abstand minimiert. Abbildung 3.3 illustriert dies.

Das gesuchte Minimum kann mithilfe von dynamischer Programmierung [8, S.323ff.] gefunden werden und zwar mithilfe der folgenden Rekurrenz:

$$\gamma(i,j) = \begin{cases} \sum_{k=0}^{j} \delta(a_0, b_k) & i = 0 \\ \sum_{k=0}^{i} \delta(a_k, b_0) & j = 0 \\ \delta(a_i,\ b_j) + \min\{\, \gamma(i-1, j-1),\ \gamma(i-1, j),\ \gamma(i, j-1) \,\} & \text{sonst} \end{cases} \tag{3.6}$$

In dieser Form hat DTW die Laufzeit- und Speicherplatzkomplexität $\mathcal{O}(nm)$. Darum wundert es nicht, dass es einige Anstrengungen gibt, den Algorithmus zu optimieren. Auch für Detexify ist dieses Laufzeitverhalten nicht ausreichend, um eine Erkennung in Echtzeit zu ermöglichen.

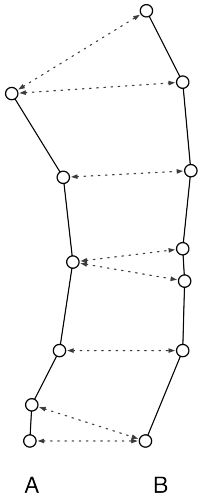


Abbildung 3.3.: Elastische Zuordnung bei DTW

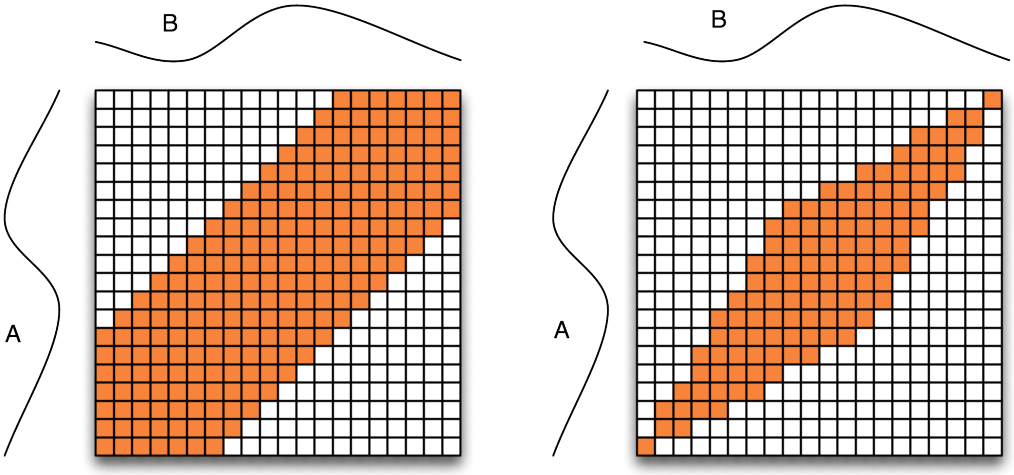


Abbildung 3.4.: Typische Beschränkungen das Warping-Pfads
Links: Sakoe-Chiba-Band, Rechts: Ikatura-Parallelogramm

3.8.2. Beschränkung des Warping-Pfads

Es ist möglich DTW dadurch zu beschleunigen, dass bei der Suche nach dem optimalen Warping-Pfad nicht die ganze Matrix durchsucht wird. Stattdessen wird der Suchraum beschränkt. Abb. 3.4 zeigt üblicherweise verwendete globale Beschränkungen. Es ist auch möglich lokale Beschränkungen wie von Rabiner u. Juang [34] beschrieben festzulegen, Keogh u. Ratanamahatana [21] bemerken aber, dass das dies äquivalent zu globalen Beschränkungen sei. Nun könnte man glauben, dass sich eine solche Einschränkung nachteilig auswirkt. Ratanamahatana u. Keogh [36] berichten aber in ihrer Arbeit, dass eher das Gegenteil der Fall ist und Beschränkungen des Pfads sogar helfen, die Erkennungsraten zu verbessern. Dies kann dadurch verstanden werden, dass eine Beschränkung pathologisches Warpen verhindert [21].

Wenn beide Folgen die gleiche Länge n haben, ist eine Möglichkeit den Warping-Pfad zu beschränken, einen Korridor um die Diagonale der Matrix festzulegen, aus dem nicht ausgebrochen werden darf. Beschrieben werden kann dies durch eine Beschränkung der Indizes des Warping-Pfads $w_k = (i, j)_k$ und zwar gilt dann

$$j - r(i) \leqslant i \leqslant j + r(i), \tag{3.7}$$

wobei $r(i)$ den erlaubten Korridor beschreibt.

Im Fall von $r(i) = s$ also r konstant reduziert sich damit die Komplexität von DTW von $\mathcal{O}(n^2)$ auf $\mathcal{O}(ns)$. Zusätzlich ermöglicht die Beschränkung bei fester Länge n der Folgen eine weitere noch effektivere Optimierung, die im folgenden Abschnitt beschrieben wird.

3.8.3. Untere Schranken

Gegeben ein Abstandsmaß $\delta : S \times S \to \mathbb{R}_0^+$ und eine untere Schranke δ_{LB} d.h.

$$\delta_{LB}(a, b) \leqslant \delta(a, b) \; \forall \; a, b \in S,$$

die deutlich günstiger zu berechnen ist, dann lässt sich mit Algorithmus 1 eine sequentielle Suche nach dem nächsten Nachbarn eines Elements $c \in S$ beschleunigen (vergl. [21]).

Für DTW für Folgen derselben Länge n mit reellen Messpunkten mit einer Beschränkung des Warping-Pfads der Form 3.7 haben Keogh u. Ratanamahatana [21] eine scharfe untere Schranke (LB_Keogh) angegeben. Zudem haben Ratanamahatana u. Keogh [36] die Kritik entkräftet, dass vor der Verwendung einer solchen Optimierung eine lineare Interpolation der Folgen nötig ist, um sie auf dieselbe Länge zu bringen, und damit der Vorteil, Folgen unterschiedlicher Länge vergleichen zu können, verlogen ginge. Eine untere Schranke scheint also eine reizvolle Möglichkeit zu sein DTW zu beschleunigen.

In Fall von Detexify sind die einzelnen Punkte der Folgen aber Punkte im $\mathbb{R}^2$ (siehe 3.4) und es ist intuitiv einzusehen, dass eine Reduktion auf eine Dimension einen zu hohen Informationsverlust nach sich zöge. Es ist leider nicht leicht im $\mathbb{R}^2$ auf eine zu [21] analoge Weise eine untere Schranke zu definieren, die sinnvoll einsetzbar ist.

Algorithmus 1 Beschleunigung der Suche nach dem nächsten Nachbarn eines Elements c durch eine untere Schranke

```
best_so_far ← ∞
for all samples s do
  lb_dist ← δ_LB(c, s)
  if lb_dist < best_so_far then
    true_dist ← δ(c, s)
    if true_dist < best_so_far then
      best_so_far ← true_dist
      nearest_neighbor ← s
    end if
  end if
end for
```

3.8.4. Eine untere Schranke für Folgen in $\mathbb{R}^2$

LB_Keogh

Bevor ich die Konstruktion einer unteren Schranke für Folgen im $\mathbb{R}^2$ erläutere, gebe ich noch eine kurze Erklärung der von Keogh u. Ratanamahatana [21] definierten unteren Schranke LB_Keogh an.

Seien

$$Q = q_1 \dots q_n \qquad q_i \in \mathbb{R} \tag{3.8}$$

$$C = c_1 \dots c_n \qquad c_i \in \mathbb{R} \tag{3.9}$$

zwei Folgen gleicher Länge n. $\mathcal{W}_r$ sei die Menge der durch r beschränkten Warping-Pfade (siehe 3.7). DTW sei (leicht verändert) gegeben durch

$$DTW(Q, C) = \min_{W \in \mathcal{W}_r} \sqrt{\sum_{i=1}^{n} d_{w_i}}. \tag{3.10}$$

Mithilfe von r werden nun weitere Folgen U und L definiert:

$$U_i = \max\{q_{i-r}, \dots, q_{i+r}\} \tag{3.11}$$

$$L_i = \min\{q_{i-r}, \dots, q_{i+r}\} \tag{3.12}$$

Es gilt offensichtlich

$$U_i \geqslant q_i \geqslant L_i \quad \forall\, i. \tag{3.13}$$

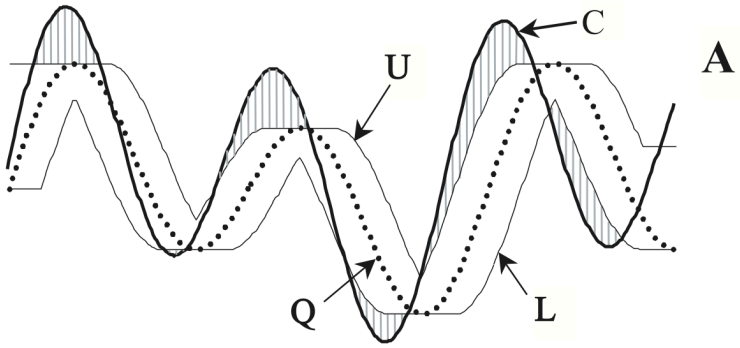


Abbildung 3.5.: Grafische Darstellung von LB_Keogh. Grafik entnommen aus [21]

Nun können wir eine untere Schranke für 3.10 wie folgt definieren:

$$\mathrm{LB\_Keogh}(Q, C) = \sqrt{\sum_{i=1}^{n} \begin{cases} \delta(c_i, U_i) & c_i > U_i \\ \delta(c_i, L_i) & c_i < L_i \\ 0 & \text{sonst} \end{cases}} \tag{3.14}$$

Die Intuition dahinter ist die Folgende: Die Folgen U und L bilden eine Hülle um die ursprüngliche Folge Q. LB_Keogh ist dann sozusagen der Abstand einer neuen Folge C zu dieser Hülle (siehe Abbildung 3.5). Dass dies tatsächlich eine untere Schranke ist, haben Keogh u. Ratanamahatana [21] gezeigt. Sie argumentieren auch, dass ihre Schranke schärfer als alle bisher in der Literatur vorgestellten unteren Schranken sei.

$\mathbb{R}^2$-analog zu LB_Keogh

Seinen nun zwei Folgen

$$A = a_1 \dots a_n \qquad a_i \in \mathbb{R}^2 \tag{3.15}$$

$$B = b_1 \dots b_n \qquad b_i \in \mathbb{R}^2 \tag{3.16}$$

gegeben. $\mathcal{W}_r$ sei wieder die Menge der durch r beschränkten Warping-Pfade. Wir definieren dann eine neue Folge C durch

$$C_i = \mathrm{conv}\,\{a_{i-r}, \dots, a_{i+r}\}, \tag{3.17}$$

wobei

$$\mathrm{conv}\, X = \bigcap_{\substack{X \subseteq K \subseteq \mathbb{R}^2 \\ K \text{ konvex}}} K$$

die konvexe Hülle ist. Die neue Folge C besteht also aus konvexen Polygonen. Für 3.10 lässt sich nun eine untere Schranke wie folgt definieren:

$$\mathrm{LB}(A,B) = \sqrt{\sum_{i=1}^{n} \begin{cases} 0 & b_i \in C_i \\ \delta(b_i, C_i) & \text{sonst} \end{cases}}. \tag{3.18}$$

Dabei ist natürlich $\delta(b_i, C_i) = \min_{c \in C_i} \delta(b_i, c)$.

Das Problem besteht nun aber darin, dass diese untere Schranke nur dann effizient zu berechnen wäre, wenn beim Bilden der konvexen Hülle viele Punkte im inneren derselben lägen, denn die Komplexität der Berechnung des Abstandes zur konvexen Hülle ist proportional zur Anzahl der Ecken v. Es gilt zwar $(v \leqslant r)$, aber es ist völlig unklar, wie stark v typischerweise unter r liegt. Geht man von einer Gleichverteilung zwischen 2 und r aus, so bleibt damit die Komplexität dieser unteren Schranke bei $\mathcal{O}(rn)$, was der des ursprünglichen Algorithmus entspricht. Dadurch ist also nichts gewonnen.

Um bei der Berechnung zu sparen lässt sich C stattdessen als

$$C_i = \text{boundingbox}\,\{a_{i-r}, \ldots, a_{i+r}\} \tag{3.19}$$

definieren wobei

$$\text{boundingbox}\,X = \bigcap_{Q=[a,b]\times[c,d] \supseteq X} Q.$$

Die Formel 3.18 für die untere Schranke mit dieser Version von C wäre dann für große Werte von r günstiger zu berechnen, aber dafür leidet die Schärfe der Schranke, was der Laufzeit von Algorithmus 1 (Seite 30) natürlich schadet.

Es gibt aber eine weitere Optimierung, die vielversprechender ist, in jedem Fall eine lineare Laufzeit hat und ohne eine Beschränkung der Warping-Pfade auskommt. Aus diesem Grund verzichte ich auf die Angabe eines formalen Beweises, dass es sich bei meinen Varianten überhaupt um untere Schranken handelt.

3.8.5. Greedy Matching

Es ist möglich DTW mit einem Greedy-Algorithmus [8, S.37off.] zu approximieren.

Grundsätzlich funktioniert eine Greedy-Approximation von DTW bei zwei Folgen A und B mit $a_i, b_i \in S$ und einem Abstandsmaß $\delta : S \times S \to \mathbb{R}_0^+$ folgendermaßen: Beginne mit den Startpunkten a_1 und b_1 und ordne diese einander zu. Ordne dann jeweils die nächsten Punkte so einander zu, dass der Abstand unter δ minimal ist (wähle das lokale Optimum). Wiederhole dies, bis in einer der beiden Folgen nur noch ein Punkt übrig ist. Ordne alle restlichen Punkte der anderen Folge diesem zu. Algorithmus 2 (Seite 33) zeigt eine Implementierung in Pseudocode. Dieser Algorithmus hat einen konstanten Speicherplatzaufwand und eine lineare Laufzeit.

Algorithmus 2 Greedy Matching

```
a ← next from A
b ← next from B
d ← δ(a, b)
a′ ← next from A
b′ ← next from B
while points left in A ∧ points left in B do
    l, m, r ← δ(a′, b), δ(a′, b′), δ(a, b′)
    μ ← min {l, m, r}
    d ← d + μ
    if l = μ then
        a ← a′
        a′ ← next from A
    else if r = μ then
        b ← b′
        b′ ← next from B
    else
        a ← a′
        b ← b′
        a′ ← next from A
        b′ ← next from B
    end if
end while
if no points left in A then
    for all points p in B do
        d ← d + δ(a′, p)
    end for
else if no points left in B then
    for all points p in A do
        d ← d + δ(b′, p)
    end for
end if
```

MacLean u. Labahn [26] stellen eine Variante dieses Algorithmus vor. Sie verwenden ihn zur Erkennung mathematischer Symbole in MathBrush, das in dieser Arbeit bereits erwähnt wurde (siehe auch 1.2.7 und [25]), gehen bei der Zuordnung von Punkten aber simultan von beiden Seiten vor. Welche Vorteile das gegenüber einer einseitigen Strategie bringt, erklären sie nicht. Sie vergleichen ihren Greedy-Algorithmus jedoch mit dem Standard-DTW-Algorithmus und kommen zum Ergebnis, dass der Greedy-Algorithmus zur Unterscheidung von mathematischen Symbolen ebenso geeignet ist. Insbesondere bei einer kleinen Anzahl von Punkten pro Strich (<30) ist der Greedy-Algorithmus sogar gleich gut geeignet und durch seine lineare Laufzeit natürlich wesentlich schneller. Bei ihren Tests verwenden sie 15796 handgeschriebene Symbole aus 70 Symbolklassen, geschrieben von 20 unterschiedlichen Personen. Ich werde diese Variante von nun an Greedy Dynamic Time Warping (GDTW) nennen.

3.8.6. Elimination

Eine weitere Optimierungsmöglichkeit besteht darin, irrelevante Trainingsmuster im Suchraum zu eliminieren, um für diese einen teuren DTW-Vergleich gar nicht mehr durchführen zu müssen. Watt u. Xie [50] schlagen Elimination durch klassische Merkmale vor, d.h. es wird ein Merkmalsvektor gebildet und nur noch für die besten N Trainingsmuster, die auf Basis eines günstigen Abstandsmaßes gefunden werden, die DTW-Entfernung ausgerechnet. Watt u. Xie [50] verlieren durch ein solches Verfahren 3% an Erkennungsgenauigkeit, aber vermeiden für fast 90% des Suchraumes das teurere Abstandsmaß. Ist das verwendete Abstandsmaß eine Metrik, so gibt es sogar exakte Eliminationsverfahren für die Nächste-Nachbarn-Klassifikation, die ohne Genauigkeitsverlust auskommen, z.B. im $\mathbb{R}^2$ mithilfe von Voronoi-Diagrammen. DTW ist leider keine Metrik, darum bleiben mir solche Mittel verschlossen. Aus diesem Grund werden Eliminationsverfahren in Detexify nicht zur Anwendung gebracht.

3.8.7. Parameter

Bei der Verwendung von template matching mit DTW als Abstandsmaß müssen einige Parameter festgelegt werden. Diese beeinflussen die Leistungsfähigkeit und Geschwindigkeit des Klassifizierers und müssen sorgfältig ausgewählt werden. Die Parameter sind die folgenden:

n - Anzahl der Punkte pro Strich

r - Fensterbreite bei beschränktem DTW (siehe 3.7)

δ - das verwendete innere Abstandsmaß (siehe 3.8.1)

k - Anzahl der nächsten Nachbarn beim klassischen k-Nächste-Nachbarn-Algorithmus

C - Anzahl der Trainingsmuster pro Klasse

Da DTW schon häufig Verwendung gefunden hat, gibt es natürlich in der Literatur einige Angaben zu experimentell gefundenen Optima für bestimmte Parameter. Dabei ist zu bedenken, dass die optimale Parameterwahl immer vom konkreten Problem und natürlich auch von den Testdaten abhängt. Trotzdem bieten die Angaben gute Anhaltspunkte, in welchen Bereichen sich sinnvolle Parameter bewegen.

Golubitsky u. Watt [16] geben an, dass bei ihrer Konfiguration mit 107 Symbolklassen die Erkennungsrate mit Steigerung von C fast linear ansteigt, bis 25 Trainingsmuster pro Klasse erreicht sind. Ab dann sind die Zuwächse nicht mehr signifikant. Sie geben das Optimum für n bei $k = 1$ mit 25 an. Bei mehr Punkten fällt die Erkennungsrate sogar leicht. Sie haben auch mit der Anzahl der nächsten Nachbarn k experimentiert und kommen zum Schluss, dass bei kleinen Klassen ($C < 10$) eine Anzahl von $k = 1$ am besten abschneidet. Werden die Klassen größer, so lohnen sich nach und nach immer größere Werte für k. Zum Beispiel ist für $C > 25$ eine Anzahl von $k = 5$ die beste Wahl. Der Vorsprung in der Erkennungsrate liegt aber bei weniger als 1%.

Golubitsky u. Watt [18] geben für $k = 1$ und beschränkte Warping-Pfade die optimalen Parameter mit $n = 30$ und $r = 2$ an. Sie verwenden dabei quadratischen euklidischen Abstand als inneres Abstandsmaß δ.

MacLean u. Labahn [26] verwenden in ihrer Greedy-Variante von DTW ein Abstandsmaß, das auch die Steigung der Striche mit einbezieht, wie auch Tappert [44] es vorgeschlagen hat. Vuong u. a. [47] gehen noch einen Schritt weiter und verwenden sogar die Krümmung der Striche.

4. Benchmarks

In diesem Kapitel stelle ich Benchmarks auf Basis der durch die Nutzer zur Verfügung gestellten Daten vor. Ich habe Benchmarks mit einem kleinen Datensatz und einem großen Datensatz durchgeführt. Der kleine Datensatz wurde verwendet, um die Parameter zu optimieren. Durch die relativ geringe Anzahl der Tests konnte so in schnellen Iterationen eine große Menge an Benchmarks durchgeführt werden. Der große Benchmark dient dazu, einen Blick auf die Performance des im Einsatz befindlichen Systems unter realen Bedingungen zu liefern, und vor allem um festzustellen, ob die in 3.5 geforderte *Skalierbarkeit* und ein ausreichend gutes *Laufzeitverhalten* realisiert werden. Dabei wird aufgrund der *Interaktivität* der Anwendung immer ein besonderes Augenmerk auf die Top 5-Erkennungsrate gelegt.

Die Daten, auf denen die Benchmarks basieren, sind die aus dem laufenden System, das unter der Adresse `http://detexify.kirelabs.org` aufgerufen werden kann. Sie haben sich über den Verlauf eines Jahres angesammelt. Wie in 2.4 bemerkt, wurden die 147.881 Trainingsmuster grob bereinigt, so dass für die Benchmarks eine Menge von 133.462 Mustern zur Verfügung stand. Die einzelnen Symbolklassen schwanken dabei stark in der Menge der Trainingsmuster. Während für einzelne Symbole 2.000 und mehr Trainingsmuster in der Datenbank sind, gibt es eine große Menge Symbole, die nur 30 und weniger besitzen. Bei der Bereinigung wurden auch nur offensichtlich falsche Zeichnungen aussortiert, und schlecht zu erkennende oder unsauber gezeichnete Muster wurden bewusst in der Datenbank belassen, um ein realistisches Bild auf die zu erwartende Erkennungsgenauigkeit unter realen Bedingungen bieten zu können und damit auch Schwächen zu finden.

4.1. Kleiner Datensatz

In diesem Abschnitt werden mithilfe eines Teils der verfügbaren Trainingsdaten die DTW-Varianten auf ihre Güte überprüft und die Parameter optimiert. Dabei wird auf die folgende Weise verfahren: Zuerst werden mit fest gewählten Parametern C, δ und n die Varianten DTW und GDTW gegeneinander getestet. Dann werden nach und nach die unterschiedlichen Parameter einzeln unter Fixierung der restlichen Parameter variiert und so ein optimaler Wert erhalten.

Der kleine Datensatz besteht aus 100 zufällig aus der Datenbank ausgewählten Symbolen. Aus den vorhandenen Trainingsmustern wurden pro Symbolklasse 75 zufällig ausgewählt, und davon wurden 50 für das Training des Servers und 25 als Testmuster verwendet. Dies ergibt 2500 Tests.

Folgende Symbole wurden verwendet:

$, {, ©, }, §, &, #, %, ✓, å, Å, æ, Ð, Đ, €, ∪, ⊕, ×, ∗, ⊗, ±, ∩, ∨, ·, ⊙, ∧, ∘, ⊗, ∏, ∑, ⊙, ∫, ∮, ≈, ≡, ⊥, ≅, ∝, ⊢, ∼, ≃, ∴, ∵, ⊆, ⩾, ⩽, ≪, ≠, ≲, ≳, ≜, ⇒, →, ⇔, ↦, α, θ, τ, β, ϑ, π, γ, ϕ, δ, ρ, φ, ϵ, λ, χ, ε, μ, σ, ψ, ζ, ν, ω, η, ξ, Γ, Λ, Σ, Ψ, Δ, Ξ, Ω, Θ, Π, Φ, ⊥, ∀, ℓ, ħ, ∈, ∉, ∂, ∃, [, /, ℵ, ∞

Davon sind manche wie ⊙ (`\odot`) und ⨀ (`\bigodot`) oder ∑ (`\sum`) und Σ (`\Sigma`), eigentlich dieselben Symbole. Es wurden aber keine Maßnahmen ergriffen, um solche Zweideutigkeiten aufzuheben.

4.1.1. DTW-Variante

Der erste Benchmark testet DTW in seiner klassischen Form gegen die Greedy-Approximation GDTW. Dabei wurden erst einmal die Parameter $C = 50$, $n = 25$ (optimal nach Golubitsky u. Watt [16]) festgelegt, und als Abstandsmaß δ die euklidische Metrik verwendet. Abbildung 4.1 illustriert die Ergebnisse. Es zeigt sich, dass GDTW kaum schlechter abschneidet als DTW und dabei, wie aus Abbildung 4.2 ersichtlich ist, wesentlich schneller ist. Dies bestätigt die Ergebnisse von MacLean u. Labahn [26], die ebenfalls eine Greedy-Variante von DTW verwenden.

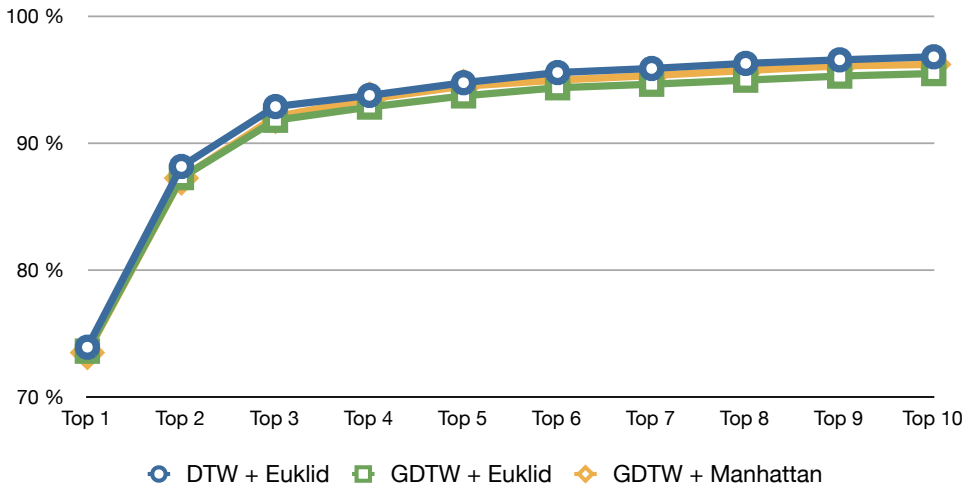


Abbildung 4.1.: Vergleich der Erkennungsraten von DTW und GDTW

Auf Basis dieser Daten fällt es leicht, die Auswahl des Verfahrens für Detexify auf GDTW festzulegen und im Folgenden die Betrachtungen auf dieses zu beschränken. DTW mit be-

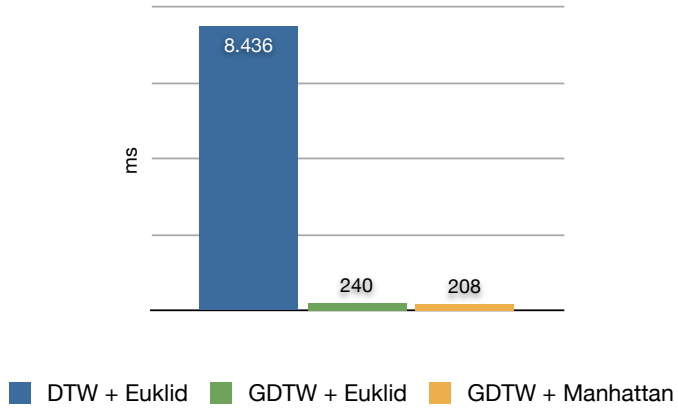


Abbildung 4.2.: Vergleich der Laufzeit von DTW und GDTW

schränkten Warping-Pfaden wurde nicht mehr betrachtet, da es nicht die Laufzeiteigenschaften von GDTW bieten kann.

4.1.2. Inneres Abstandsmaß

Abbildung 4.1 zeigt neben den Erkennungsraten von DTW und GDTW mit euklidischer Metrik als Abstandsmaß auch noch die Erkennungsraten von GDTW mit der Manhattan-Metrik als Abstandsmaß, die als

$$d(x, y) = \sum_i a_i - b_i$$

definiert ist. Dieser Benchmark wurde wieder mit den Parametern $C = 50$ und $n = 25$ durchgeführt. Die Manhattan-Metrik ist nicht nur günstiger in der Berechnung, sondern liefert auch noch eine leichte Verbesserung der Top 5-Erkennungsrate um 0,6%. Die folgenden Benchmarks wurden daher alle mit der Manhattan-Metrik d als innerem Abstandsmaß δ durchgeführt.

4.1.3. Anzahl Trainingsmuster

Die Anzahl der verwendeten Trainingsmuster C hat natürlich einen Einfluss auf die Erkennungsraten.

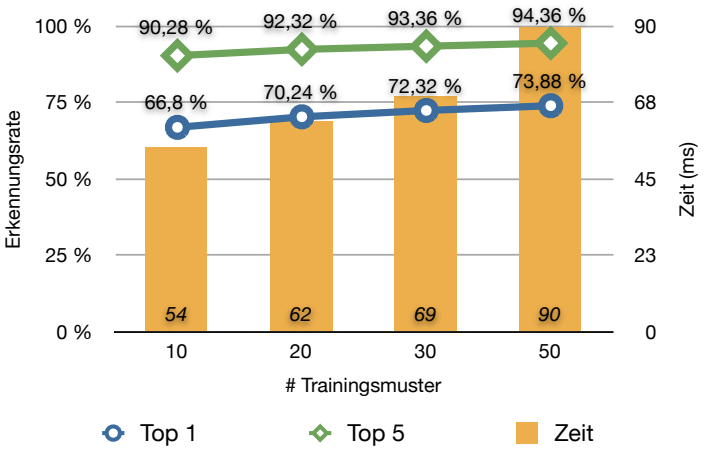


Abbildung 4.3.: Einfluss der Anzahl der Trainingsmuster C

Abbildung 4.3 zeigt, dass, anders als Golubitsky u. Watt [16] für ihre Tests beschreiben, die Erkennungsraten (insbesondere Top 1) für $C > 25$ noch deutlich steigen (weiterhin $n = 25$). Dies liegt vermutlich an einer höheren Variation in meinen Trainingsdaten, so dass sich eine größere Datenbasis auszahlt. Die Wahl von $C = 50$ wurde jedoch beibehalten und nicht noch weiter erhöht, denn C wirkt sich linear auf die Laufzeit des Erkennungsvorgangs aus.

4.1.4. Anzahl Punkte pro Strich

Die Anzahl der Punkte pro Strich n hat ebenso sowohl einen Einfluss auf die Erkennungsrate als auch auf die Laufzeit. Abbildung 4.4 zeigt, dass mit $C = 50$ ab $n = 10$ gerade die Top 1-Erkennungsrate mit sinkendem n stark abnimmt. Die Top 5-Erkennungsrate bleibt jedoch recht stabil. Für die weiteren Tests wurde $n = 10$ als optimal festgelegt, da dies nur geringe Einbußen in der Erkennung bei einer guten Laufzeit liefert.

4.1.5. Dominante Punkte

Alle bisherigen Tests wurden ohne die Filterung von Punkten mit geringer Krümmung vorgenommen. In diesem Benchmark wird diese Filterung mit unterschiedlichen Grenzwinkeln α nach der äquidistanten Verteilung der Punkte durchgeführt. Die Parameter $C = 50$, $n = 10$ und $\delta = d$ bleiben dabei fix. Abbildung 4.5 zeigt, dass beim optimalen Grenzwinkel $\alpha = 15°$ die Erkennungsraten von dieser Maßnahme stark profitieren. Die Top

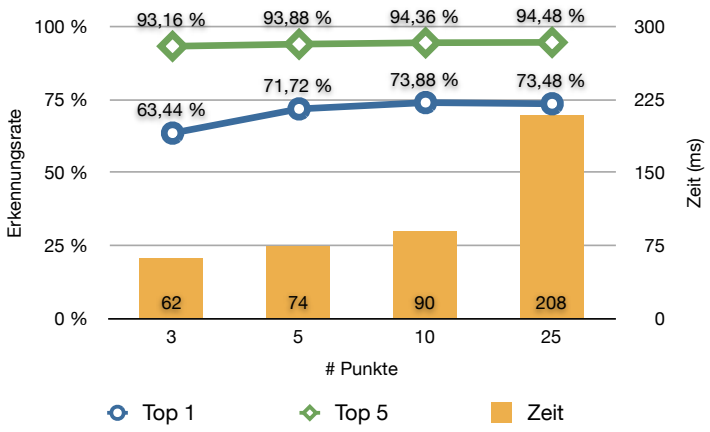


Abbildung 4.4.: Einfluss der Anzahl der Punkte pro Strich n

1-Erkennungsrate steigt um 1,48% und die Top 5-Erkennungsrate sogar um 1.6%.

4.2. Großer Datensatz

Der Benchmark mit dem großen Datensatz wurde mit den im vorherigen Abschnitt experimentell gefundenen optimalen Parametern $C = 50$, $n = 10$, $\delta = d$ und $\alpha = 15°$ durchgeführt. Dafür wurden diejenigen Symbole aus der Datenbank ausgewählt, die mindestens 15 Trainingsmuster hatten. Von diesen wurden bis zu 100 Trainingsmuster selektiert, und mit zwei Dritteln davon wurde jeweils der Server trainiert und der Rest als Testmuster verwendet.

Der Benchmark umfasst damit 627 Symbole. Dies ist eine weit höhere Symbolklassenanzahl als in allen mir bekannten Veröffentlichungen über Symbolerkennung. 27460 Trainingsmuster stehen 13411 Testmustern gegenüber.

Abbildung 4.6 zeigt die Ergebnisse. Ein einzelnder Test mit diesem Datensatz dauerte 302 ms auf einem Intel Core 2 Duo 2GHz mit 4GB Ram bei Verwendung beider Kerne.

4.3. Ergebnis

Auf Basis der in den vorigen Abschnitten durchgeführten Benchmarks habe ich als optimales Verfahren *GDTW* (4.1.1) mit den Parametern

- $\delta = d$ siehe 4.1.2

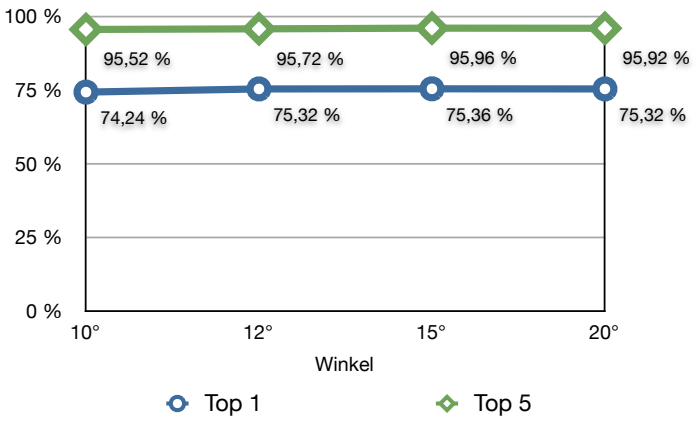


Abbildung 4.5.: Einfluss des Winkels α

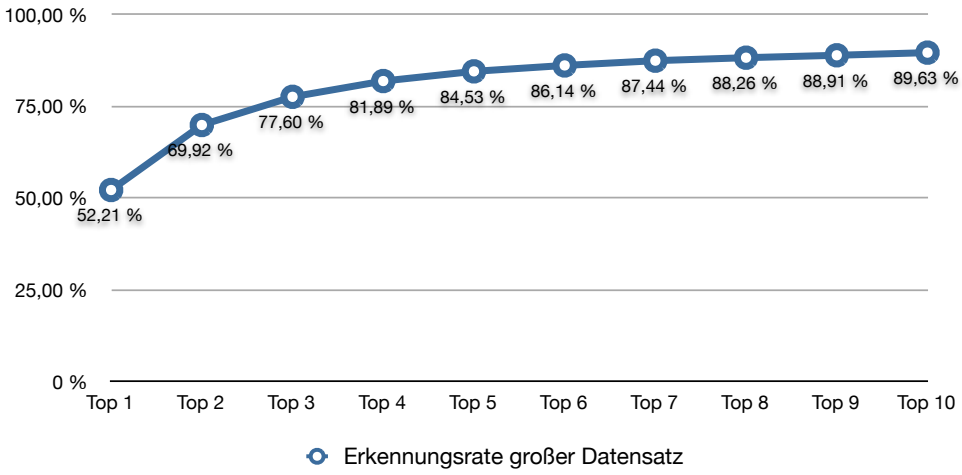


Abbildung 4.6.: Ergebnisse mit dem großen Datensatz

- $C = 50$ siehe 4.1.3
- $n = 10$ siehe 4.1.4
- $\alpha = 15°$ siehe 4.1.5

festgestellt. Dieses liefert, wie der Benchmark mit dem großen Datensatz in 4.2 zeigt, ausreichend gute *Laufzeiteigenschaften*.

Die *Skalierbarkeit* könnte besser sein. Immerhin ist der Genauigkeitsverlust beim Schritt von 100 Symbolklassen auf 627 Symbolklassen bei der Top 5-Erkennungsrate mit 11,43% nicht gerade gering. Es ist aber zu bedenken, dass sich die Anzahl der ähnlichen Symbole wie `\Updelta` (Δ) und `\Delta` (Δ) im großen Datensatz noch mal vervielfacht.

Eine weitere Erklärung der nicht überrageden Top5-Erkennungsrate von 84,52% kann die in den Tabellen 4.1 und 4.2 dargestellte Statistik aus dem Testprotokoll geben. Es ist so, dass die besterkannten Symbole ($\cong$, $\ncong$, &, $\supset$, etc.) erstens relativ eindeutige Symbole sind und andererseits über eine große Trainingsdatenmenge verfügen. Die am schlechtesten erkannten Symbole hingegen haben allesamt eine sehr kleine Trainingsmenge (<20) und sind häufig sehr ähnlich zu mindestens einem anderen Symbol, das eine wesentlich bessere Trainingsbasis hat (teilweise >1000), wie z.B. `\Updelta` (Δ) und `\Delta` (Δ) oder `\Rbag` ($\wr$) und `\int` ($\int$). Zwar werden maximal $C = 50$ Trainingsmuster je Klasse verwendet, aber es ist eine bekannte (und offensichtliche) Schwäche der Nächste-Nachbarn-Klassifikation, dass Klassen mit einer höheren Anzahl an Trainingsmustern bevorzugt ausgewählt werden, da die Wahrscheinlichkeit, dass der nächste Nachbar aus einer dieser Klassen kommt, alleine schon aufgrund der Häufigkeit höher sein muss.

Ich stelle also fest, dass die in 3.5 gestellten Anforderungen erfüllt sind. Dies bestätigt auch der subjektive Eindruck bei der Benutzung des Systems.

Tabelle 4.1.: Die 10 am besten erkannten Symbole

Befehl	Paket	Erkennungsrate
`\cong`	latex2e	93,94%
`\not\equiv`	latex2e	93,94%
`\&`	latex2e	93,94%
`\supset`	latex2e	93,94%
`\asymp`	latex2e	90,91%
`\approxeq`	amssymb	90,91%
`\neq`	amssymb	90,91%
`\gtrsim`	amssymb	90,91%
`\rightharpoonup`	latex2e	90,91%
`\circledR`	amssymb	90,91%

Tabelle 4.2.: Die 10 am schlechtesten erkannten Symbole

Befehl	Paket	Erkennungsrate
\ddag	latex2e	0,0%
\dotso	amsmath	0,0%
\Updelta	upgreek	0,0%
\ldotp	latex2e	0,0%
\dotsi	amsmath	0,0%
\Rbag	stmaryrd	0,0%
\dots	latex2e	0,0%
\Uppi	upgreek	0,0%
\Gemini	marvosym	0,0%
\therefore	amssymb	0,0%

5. Zusammenfassung und Ausblick

5.1. Zusammenfassung

In dieser Arbeit habe ich *Detexify*, ein Werkzeug zur Erkennung handgeschriebener LaTeX-Symbole, motiviert und beschrieben. Ich habe die Benutzeroberfläche und die Architektur erklärt. Ich habe mögliche Erkennungsalgorithmen erläutert und auf ihre Tauglichkeit für die Symbolerkennung in Detexify untersucht. Ich habe *template matching* mit *dynamic time warping* als bestes Verfahren für die Klassifizierung identifiziert und verschiedene Varianten und Optimierungen erörtert. Diese habe ich auf der Grundlage von Benchmarks, die auf den Daten des laufenden Systems basieren, verglichen und und ihre Parameter optimiert.

Das Ergebnis ist ein durch einen Webbrowser zugängliches Werkzeug, das den Nutzer bei der Suche nach LaTeX-Symbolen unterstützt und nicht in der Wahl des Editors einschränkt. Es steht nun schon seit einem Jahr unter der Internetadresse `http://detexify.kirelabs.org` zur Verfügung und wird aktiv genutzt (siehe auch Anhang B). Die Erfolgsraten bei der Suche sind subjektiv gut. Die Benchmarks deuten aber an, dass noch einige Luft nach oben ist.

5.2. Ausblick

Die Benchmarks haben gezeigt, dass die Erkennungsraten gerade bei vielen Symbolklassen (>600) noch nicht ideal sind. Ein großer Teil des Problems sind aber schlechte Trainingsdaten, denn hier hat sich das Crowdsourcing des Trainings (siehe 2.4) sowohl als Segen als auch als Fluch herausgestellt. Einerseits habe ich eine riesige Datenbank an Trainingsmustern erhalten, andererseits sind diese teilweise von zweifelhaftem Nutzen (siehe auch Anhang E). Hier wäre ein Bewertungssystem für Trainingsdaten denkbar, das wiederum durch die Nutzer von Detexify selbst verwaltet wird. Gleichzeitig müsste es ein Anreizsystem geben, das dazu motiviert schlecht trainierte Symbole aufzutrainieren. Auf diese Weise könnten die schlechten Daten vom Kollektiv aussortiert werden und auf lange Sicht eine solide Datenbasis geschaffen werden.

Die Optimierung des Erkennungsalgorithmus selbst ist ebenfalls noch nicht ausgereizt. Es sind einerseits noch weitere innere Abstandsmaße neben der euklidischen und Manhattan-Metrik für GDTW denkbar (siehe 3.8.7 und 4.1.1), und es wurde auch noch nicht das komplette Repertoire an in der Literatur vorgestellten Vorverarbeitungsmethoden angewandt. Ebenso interessant ist die Möglichkeit der Kombination von verschiedenen Klassifizierern. Insgesamt ist die aktuelle Implementierung aber ein zufriedenstellender Anfang.

Eine interessante Perspektive für Detexify wäre es, ein öffentliches Application Programming Interface (API) für die Symbolerkennung zur Verfügung zu stellen, so dass andere Anwendungen Detexify als Service nutzen könnten. Es wäre zum Beispiel denkbar, dass eine Anwendung Formelerkennung ermöglicht und für die Erkennung einzelner mathematischer Symbole auf Detexify zurückgreift.

Die verwendete Architektur bietet es auch an, durch mehrere Serverinstanzen und ein Loadbalancing mehrere parallele Erkennungsanfrangen gleichzeitig abarbeiten zu können. Dies wäre definitiv für eine Verwendung als Service notwendig, aber auch jetzt könnte dies die Erkennung zu Stoßzeiten beschleunigen.

Als ein völlig anderer Ansatz wäre es auch denkbar, die Erkennungsalgorithmen auf die Client-Seite zu verschieben und trotzdem eine gemeinsame Haltung der Trainingsdaten fortzuführen. Mit Hilfe von gecachten Trainingsdaten könnte die Anwendung dann auch offline arbeiten, und bei verfügbarer Verbindung würden die Trainingsdaten mit dem Stand des Online-Servers synchronisiert.

Es bestehen auf jeden Fall viele interessante Möglichkeiten auf der aktuellen Arbeit aufzubauen.

A. Quelltexte

Die Quelltexte der beiden Komponenten von Detexify (Webanwendung und Server) sind dieser Arbeit als CD beigelegt. Benutzungshinweise befinden sich in der README-Datei.

B. Nutzungsstatistiken

Dieses Kapitel enthält einige Diagramme, die Aufschluss über die Benutzer von Detexify geben können. Die Daten stammen aus dem Zeitraum vom 22.8.2010 bis zum 21.9.2010. Die Nutzungsdaten werden permanent mithilfe von Google Analytics [1] gesammelt.

Abbildung B.1.: Besucherzahlen innerhalb eines Monats. Es ist klar ein Wochenrhythmus erkennbar.

	Betriebssystem	Besuche	Anteil
1.	Windows	**9.915**	42,05 %
2.	Macintosh	**7.332**	31,10 %
3.	Linux	**6.075**	25,76 %
	Rest		1,09 %

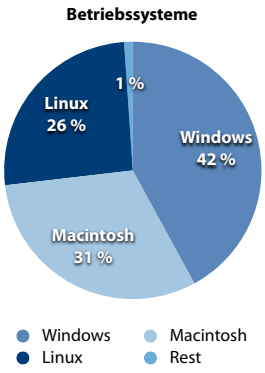


Abbildung B.2.: Betriebssysteme der Nutzer

	Browser	Besuche	Anteil
1.	Firefox	**11.568**	49,06 %
2.	Chrome	**4.697**	19,92 %
3.	Safari	**4.686**	19,87 %
4.	Internet Explorer	**1.488**	6,31 %
5.	Opera	**798**	3,38 %
	Rest		1,46 %

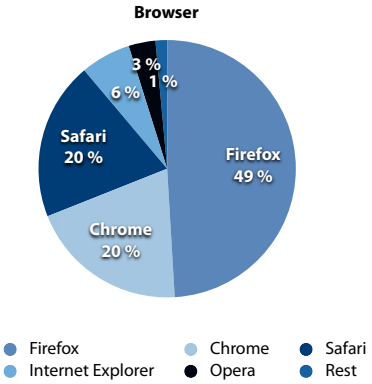


Abbildung B.3.: Webbrowser der Nutzer

	Land/Gebiet	Anteil	Anteil
1.	United States	**7.143**	30,29 %
2.	Germany	**3.417**	14,49 %
3.	France	**1.459**	6,19 %
4.	United Kingdom	**1.374**	5,83 %
5.	Taiwan	**821**	3,48 %
6.	Brazil	**770**	3,27 %
7.	Italy	**749**	3,18 %
8.	Spain	**695**	2,95 %
9.	Australia	**644**	2,73 %
10.	Netherlands	**590**	2,50 %
11.	Sweden	**567**	2,40 %
12.	Canada	**545**	2,31 %
13.	Switzerland	**499**	2,12 %
14.	China	**288**	1,22 %
15.	Denmark	**269**	1,14 %
16.	Mexico	**266**	1,13 %
17.	Argentina	**251**	1,06 %
18.	Chile	**246**	1,04 %
19.	Japan	**244**	1,03 %
20.	Austria	**241**	1,02 %
	Rest		10,62 %

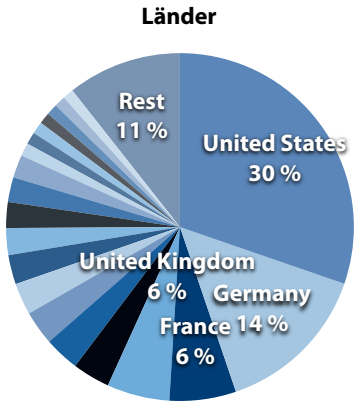


Abbildung B.4.: Länder

	Stadt	Besuche	Anteil
1.	San-Ch'Ung	**636**	2,70 %
2.	Cambridge	**616**	2,61 %
3.	Pittsburgh	**455**	1,93 %
4.	London	**441**	1,87 %
5.	New Haven	**328**	1,39 %
6.	Munich	**322**	1,37 %
7.	Paris	**301**	1,28 %
8.	Goteborg	**231**	0,98 %
9.	Bremen	**210**	0,89 %
10.	Perth	**193**	0,82 %
11.	Ithaca	**192**	0,81 %
12.	Aachen	**188**	0,80 %
13.	Zurich	**187**	0,79 %
14.	New York	**184**	0,78 %
15.	Utrecht	**173**	0,73 %
16.	Sydney	**168**	0,71 %
17.	Mexico City	**167**	0,71 %
18.	Madrid	**165**	0,70 %
19.	Amsterdam	**154**	0,65 %
20.	Berlin	**153**	0,65 %
21.	Münster	**151**	0,64 %
22.	Melbourne	**148**	0,63 %
23.	Sao Paulo	**146**	0,62 %
24.	Buenos Aires	**145**	0,61 %
25.	Berkeley	**134**	0,57 %
	Rest		73,76 %

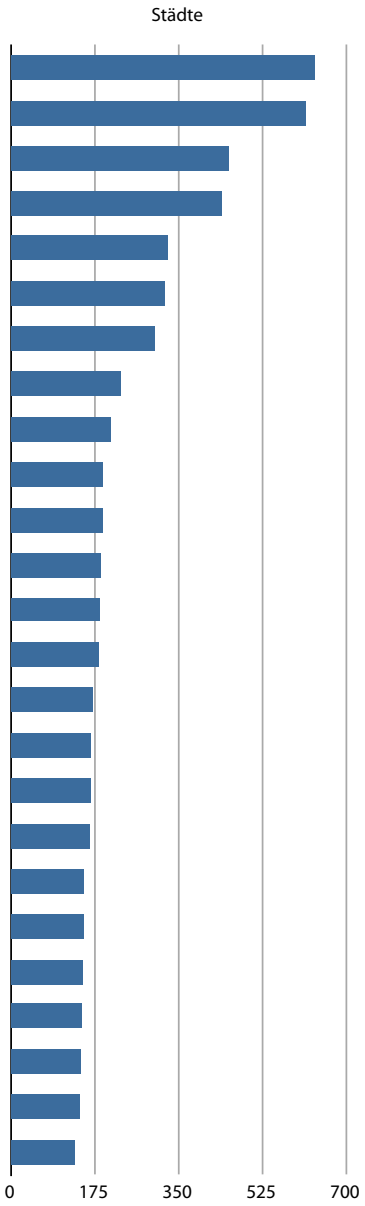


Abbildung B.5.: Städte

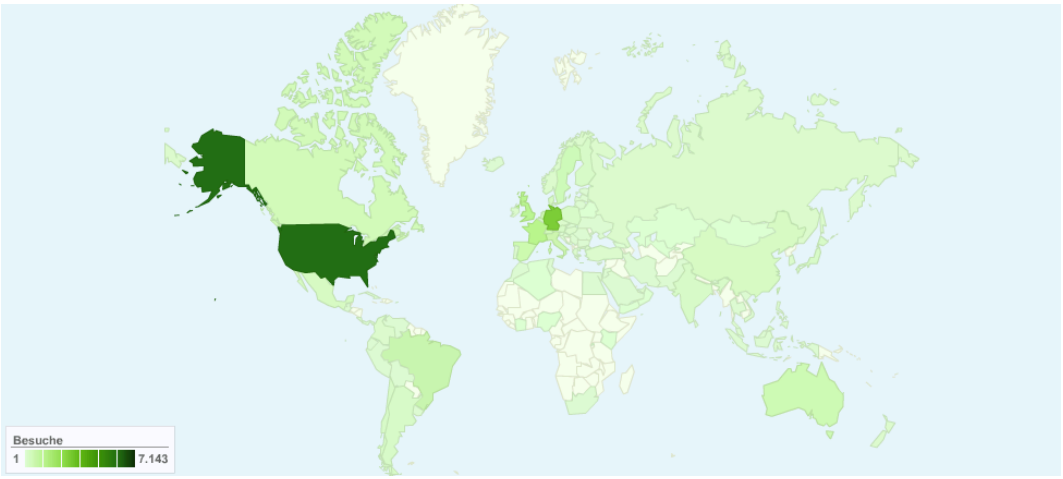


Abbildung B.6.: Karte der Zugriffe: Länder

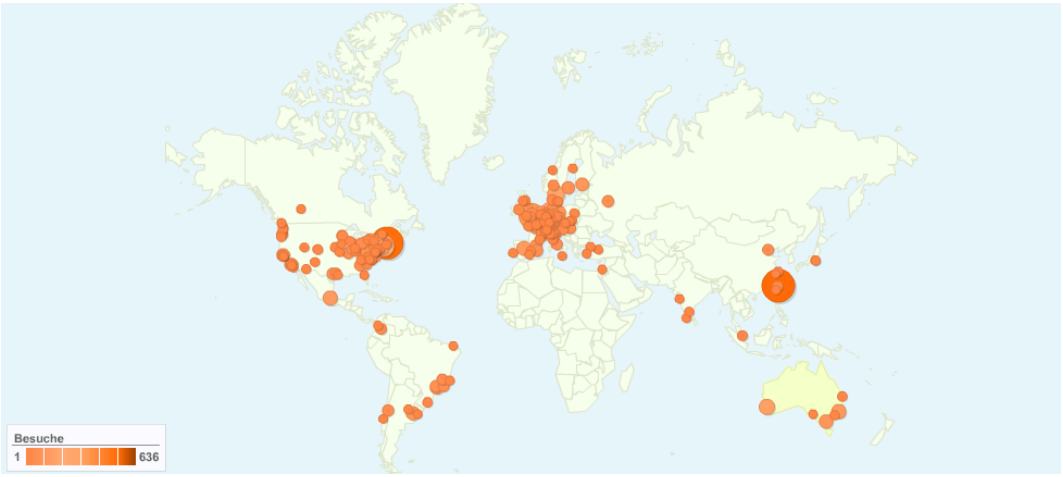


Abbildung B.7.: Karte der Zugriffe: Städte

C. Abkürzungen

API	Application Programming Interface
CAS	Computer Algebra System
DAGSVM	directed acyclic graph-SVM
DTW	Dynamic Time Warping
GDTW	Greedy Dynamic Time Warping
HMM	Hidden Markov Model
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MWV	Max-wins voting
REST	Representational State Transfer
SVG	Scalable Vector Graphics
SVM	Support Vector Machines
URL	Uniform Resource Locator
WTA	Winner-takes-all
WYSIWYG	What You See Is What You Get
WYSIWYM	What You See Is What You Mean

D. Benutzerhandbuch

In diesem Kapitel wird die Benutzung von Detexify erläutert. Die Anwendung wird aufgerufen indem man die Adresse `http://detexify.kirelabs.org` besucht.

D.1. Symbolssuche

Die erste Ansicht, nachdem man die Anwendung aufgerufen hat, ist die Symbolsuche. Der schwarz umrahmte quadratische Bereich mit dem Hinweis „`Draw here!`" ist die Zeichenfläche. Auf dieser kann ein Symbol gezeichnet werden, und nach jedem abgeschlossenen Strich beginnt der Erkennungsvorgang. Ein Indikator in der rechten unteren Ecke der Zeichenfläche weist auf einen aktiven Erkennungsvorgang hin. Währenddessen können weitere Striche auf die Zeichenfläche gezeichnet werden. Mit einen Klick auf „`clear`" in der unteren linken Ecke der Zeichenfläche kann der Erkennungsvorgang abgebrochen werden, und der Inhalt der Zeichenfläche wird gelöscht.

Erkennung

Ist der Erkennungsvorgang abgeschlossen, so erscheint rechts neben der Zeichenfläche die Ergebnisliste. Diese enthält für jedes Ergebnis eine Grafik des Symbols, den Befehl und gegebenenfalls das Paket, das benötigt wird. Es werden zuerst nur die besten fünf Ergebnisse angezeigt. Am unteren Ende der Ergebnisliste kann mit einem Klick die vollständige Ergebnisliste angezeigt werden.

Ergebnisliste

Nun kann noch mit einen Klick auf die Grafik eines Ergebnisses das entsprechende Symbol mit der aktuellen Zeichnung trainiert werden.

Training

Oberhalb der Zeichenfläche befindet sich die Navigationsleiste. Über diese kann man zur Symboltabelle gelangen.

Navigation

D.2. Symboltabelle

Über den Besuch der Adresse `http://detexify.kirelabs.org/symbols.html` oder über die Navigationsleiste durch einen Klick auf „`symbols`" gelangt man zur Symboltabelle. Diese befindet sich dann auf der linken Seite der Ansicht und ähnelt der Ergebnisliste bei der Symbolsuche.

Am oberen Rand der Tabelle befinden sich Bedienelemente, die es ermöglichen die Symboltabelle nach Befehlen oder Paketen zu filtern. Dazu muss der gewünschte Filter ausgewählt werden und eine beliebige Zeichenfolge, nach der gefiltert werden soll, in das vorhandene Textfeld eingegeben werden.

Filter

Training

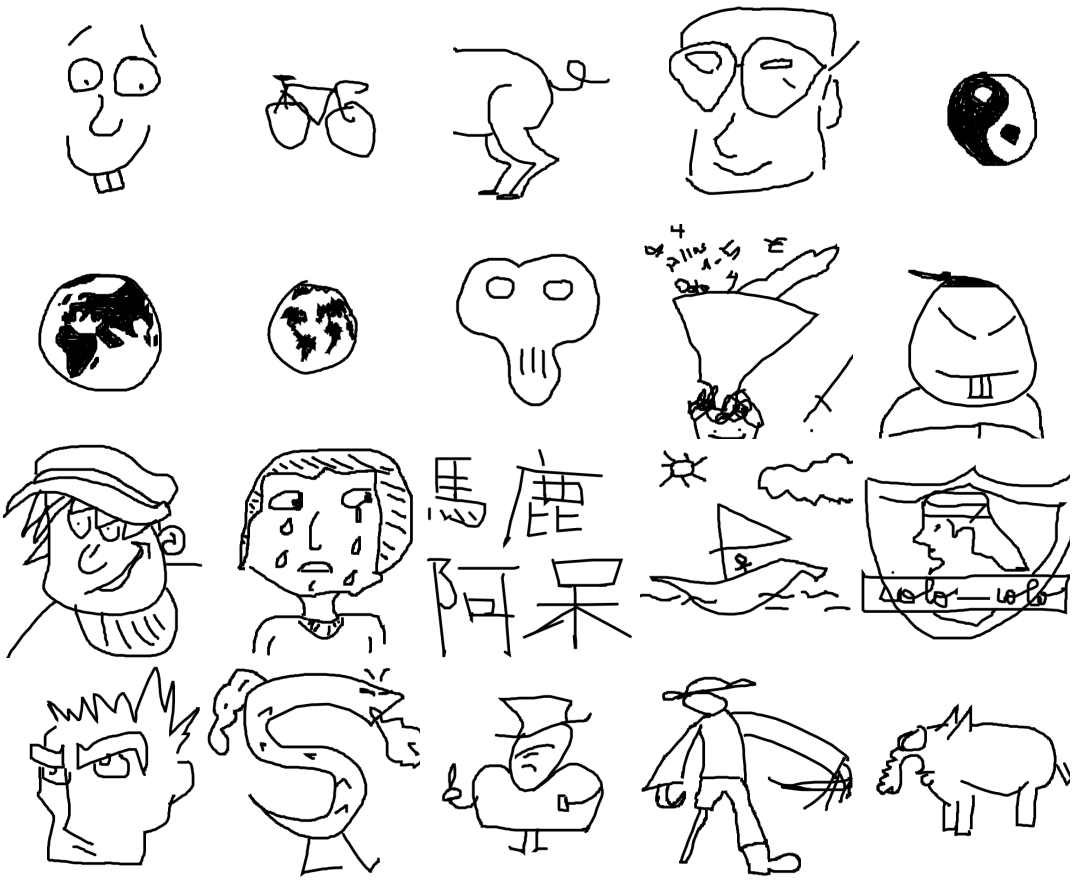
Klickt man auf einen Eintrag der Symboltabelle, so öffnet sich unterhalb dieses Eintrags eine Zeichenfläche. Auf diese kann dann das Symbol gezeichnet werden, und mit einem Klick auf „`train`" am unteren linken Rand der Zeichenfläche wird das Symbol mit der Zeichnung trainiert. Die Zeichnung kann auch ohne das Symbol zu trainieren wieder gelöscht werden, indem daneben auf „`cancel`" geklickt wird.

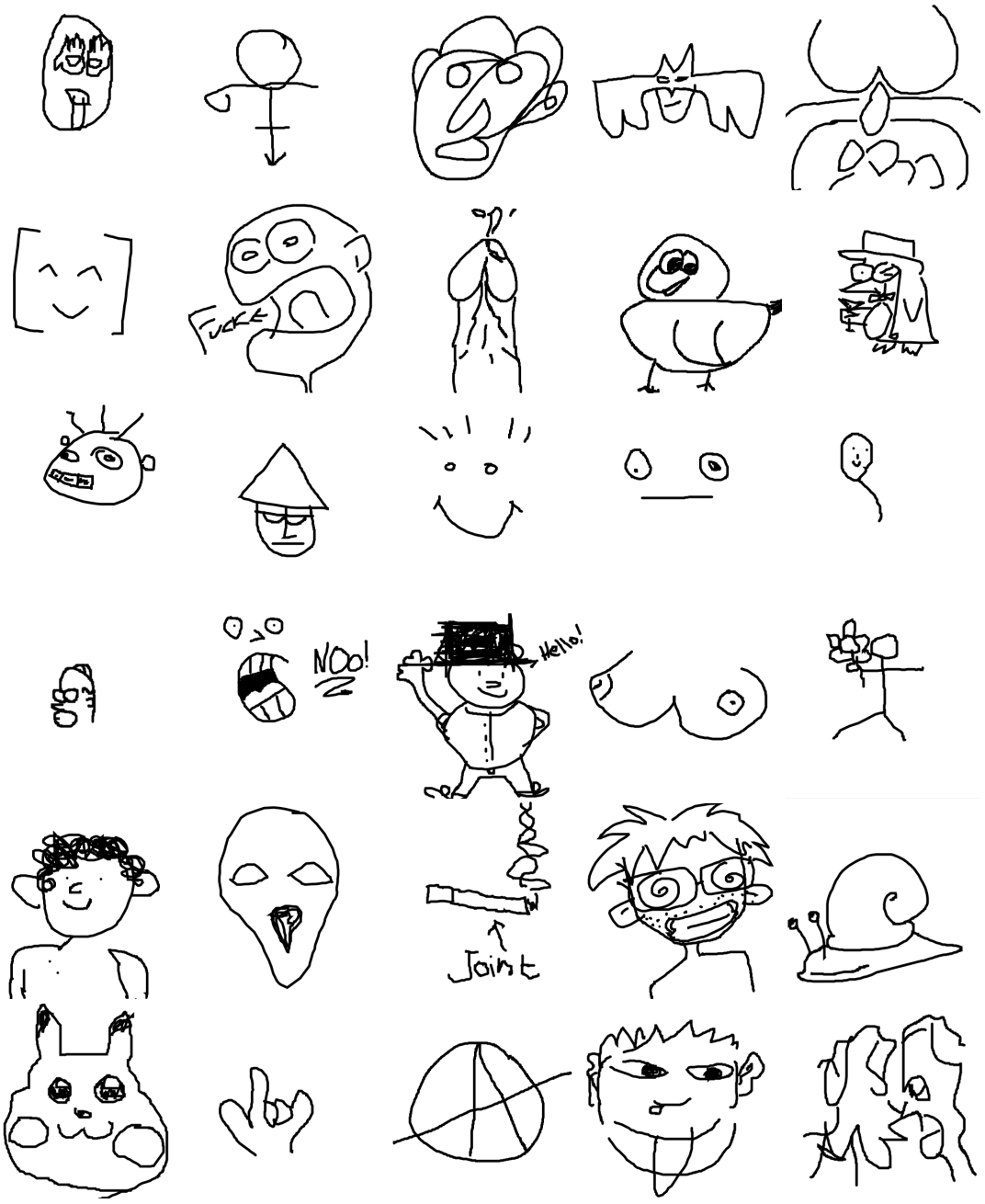
Navigation

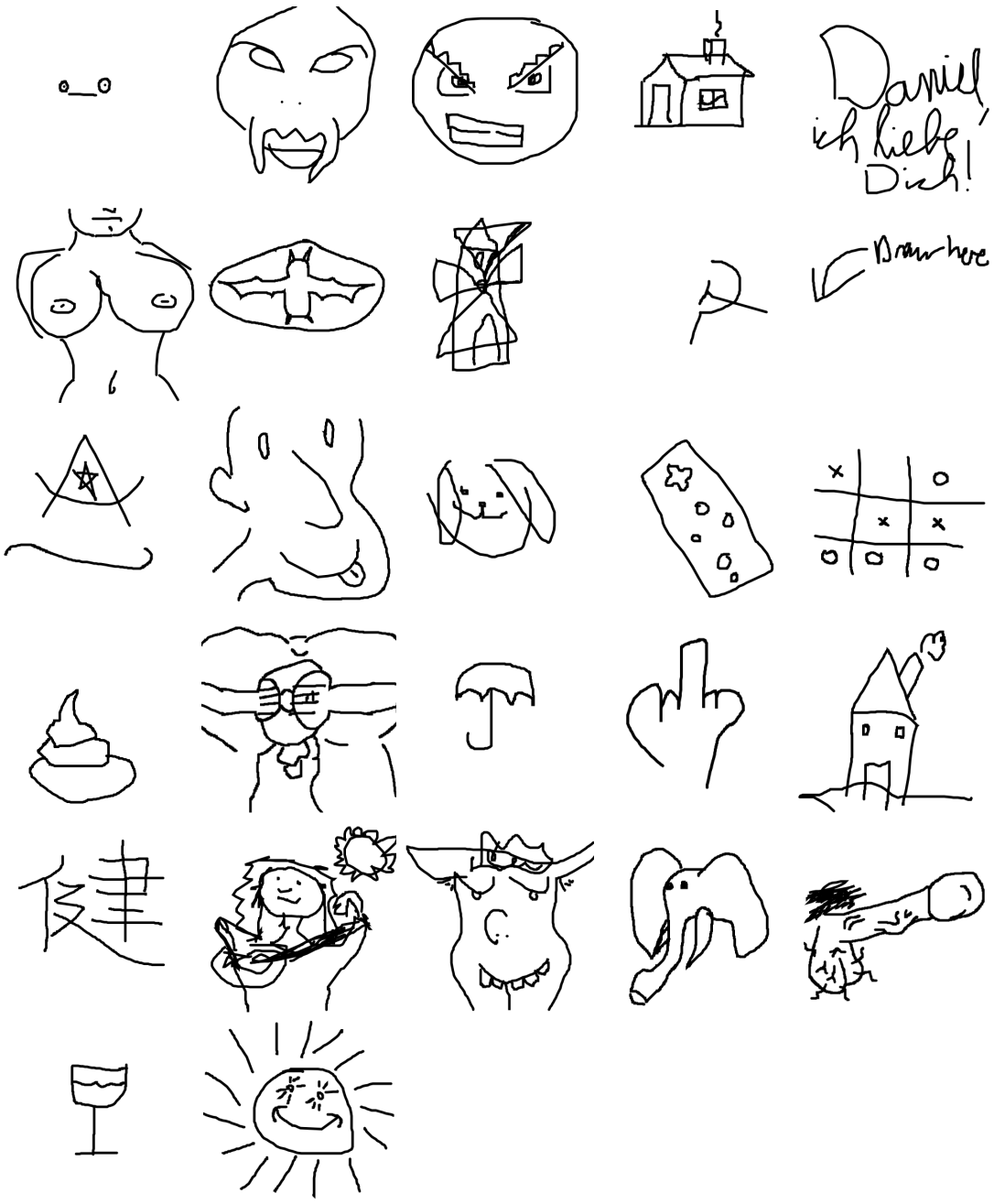
Oberhalb der Filterkontrollen befindet sich wieder die Navigationsleiste. Über diese kann man durch einen Klick auf „`classify`" zurück zur Symbolsuche gelangen.

E. Kunst

Da das Training von Detexify komplett von den Nutzern übernommen wird (siehe 2.4) und nicht überwacht wird, sammeln sich in der Datenbank mit Trainingsbeispielen allerlei Zeichnungen an, die mit dem Symbol, für das sie gespeichert, wurden nichts zu tun haben. Die interessantesten davon sind in der folgenden Übersicht aufgelistet. Dies dient einzig und allein der Belustigung des Lesers.




FUCK
NOo!
Hello!
Joint


Daniel,
ich liebe
Dich!
Draw here

Danksagungen

Ich möchte als erstes Philipp Kühl für die Idee für Detexify danken. Ohne ihn hätte ich das Projekt nie angefangen.

Desweiteren möchte ich Prof. Dr. Xiaoyi Jiang für die kompetente Betreuung (während der Diplomarbeit) und gute Hinweise in der Anfangsphase des Projekts (noch bevor es meine Diplomarbeit war) danken.

Ich danke auch Mojca Miklavec der mir in der tug-summer-of-code Mailing-Liste wertvolles Feedback gegeben hat, als ich die ersten Alphaversionen von Detexify online hatte.

Weiter gilt mein Dank Scott Pakin der „handwriting-based symbol search" als Google Summer of Code-Projekt vorgeschlagen hatte, schon bevor ich mit Detexify begonnen hatte, was ich zu diesem Zeitpunkt aber noch nicht wusste. Auch er hat mir zahlreiche helfende Hinweise in der tug-summer-of-code Mailingliste gegeben.

Ich möchte auch der Zweitag GmbH danken, in deren Räumen diese Diplomarbeit entstanden ist.

Ich danke den KorrekturleserInnen Wiebke Heep, Thomas Hollstegge, Philipp Kühl, Christian Peters und Matthias Redecker.

Außerdem danke ich den Nutzern von Detexify, insbesondere denen, die Trainingsmustern gespendet haben, und denen die Geld gespendet haben, und damit die Hostingkosten für mich niedrig gehalten haben.

Ganz besonders will ich auch den Künstlern danken, die für Anhang E verantwortlich sind.

Und zum Schluss möchte ich meinen Athleten beim TuS Hiltrup und meiner Leichtathletik-Trainingsgruppe danken, die ich leider während der Arbeit an diesem Werk etwas vernachlässigen musste. Sie sorgen für die nötige Balance in meinem Leben.

Literaturverzeichnis

[1] *Google Analytics*. http://www.google.com/analytics

[2] *JEquation*. http://jequation.sourceforge.net/

[3] *Maple*. http://www.maplesoft.com/

[4] *Stack Overflow: Best LaTeX editor for Windows*. http://stackoverflow.com/questions/270121/best-latex-editor-for-windows, Abruf: 9-9-2010

[5] *Stack Overflow: What LaTeX Editor do you suggest for Linux?* http://stackoverflow.com/questions/1235043/what-latex-editor-do-you-suggest-for-linux, Abruf: 9-9-2010

[6] Andersen, Chris: *The Long Tail*. HANSER, 2006

[7] Chan, K ; Yeung, D: Mathematical expression recognition: a survey. In: *International Journal on Document Analysis and Recognition* 3 (2000), Nr. 1, 3-15. http://www.springerlink.com/index/NGKR9789L4JNU7GY.pdf

[8] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to algorithms*. Second Edition. The MIT Press, 2001

[9] Crockford, Douglas: *JSON*. http://www.json.org/

[10] Duan, KB ; Keerthi, SS: Which is the best multiclass SVM method? An empirical study. In: Oza, Nikunj C. (Hrsg.) ; Polikar, Robi (Hrsg.) ; Kittler, Josef (Hrsg.) ; Roli, Fabio (Hrsg.): *Multiple Classifier Systems: : 6th International Workshop*, Springer, 2005, S. 278–285

[11] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, O. ; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1*. http://www.w3.org/Protocols/rfc2616/rfc2616.html

[12] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Diss., 2000

[13] Fitzgerald, J ; Huang, B ; Kechadi, T: An efficient hybrid approach for online recognition of handwritten symbols. In: Gelbukh, Alexander (Hrsg.) ; Terashima, Hugo (Hrsg.): *MICAI 2005: Advances in Artificial Intelligence*, Springer, 2005, 843–853

[14] Fitzgerald, J.A. ; Geiselbrechtinger, F. ; Kechadi, T.: Application of fuzzy logic to online recognition of handwritten symbols. In: *Proceedings of the 9th International Workshop on Frontiers in Handwritten Recognition*, 2004, 395–400

[15] Golubitsky, O. ; Watt, S.: Confidence Measures in Recognizing Handwritten Mathematical Symbols. In: *Intelligent Computer Mathematics* (2009), S. 460–466

[16] Golubitsky, O ; Watt, SM: Online computation of similarity between handwritten characters. In: *Proc. Docum. Rec and Retrieval (DRR XVI)* 1 (2009)

[17] Golubitsky, O ; Watt, SM: Online Recognition of Multi-Stroke Symbols with Orthogonal Series. In: *2009 10th International Conference on Document Analysis and Recognition* IEEE, 2009, S. 1265–1269

[18] Golubitsky, O ; Watt, SM: Distance-based classification of handwritten symbols. In: *International Journal on Document Analysis and Recognition* 13 (2010), Nr. 2, S. 133–146

[19] Howe, J: The rise of crowdsourcing. In: *Wired Magazine* 14 (2006), Nr. 6, S. 1–4

[20] Jaeger, S ; Liu, CL ; Nakagawa, M: The state of the art in Japanese online handwriting recognition compared to techniques in western handwriting recognition. In: *International Journal on Document Analysis and Recognition* 6 (2003), Nr. 2, S. 75–88

[21] Keogh, E ; Ratanamahatana, C: Exact indexing of dynamic time warping. In: *Knowledge and Information Systems* 7 (2005), Nr. 3, 358–386. http://www.springerlink.com/index/M4XVNJNM1RXW041W.pdf

[22] Keshari, B ; Watt, SM: Online mathematical symbol recognition using svms with features from functional approximation. In: *Proc. Mathematical User-Interfaces Workshop (MathUI)* Citeseer, 2008

[23] Koerich, A.L. ; Sabourin, R. ; Suen, C.Y.: Large vocabulary off-line handwriting recognition: A survey. In: *Pattern Analysis & Applications* 6 (2003), Nr. 2, 97–121. http://www.springerlink.com/index/2WJ0BA1JTXJA18K3.pdf

[24] Kosmala, A. ; Rigoll, G.: On-line handwritten formula recognition using statistical methods. In: *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on* Bd. 2 IEEE, 2002, 1306–1308

[25] Labahn, G. ; Lank, E. ; MacLean, S. ; Marzouk, M. ; Tausky, D.: Mathbrush: A system for doing math on pen-based devices. In: Kise, Koichi (Hrsg.) ; Sako, Hiroshi (Hrsg.): *Document Analysis Systems, 2008. DAS'08. The Eighth IAPR International Workshop on*, S. 599–606

[26] MacLean, S ; Labahn, G: Elastic matching in linear time and constant space. In: *Proc., Ninth IAPR Int'l. Workshop on Document Analysis Systems*, 2010, 551–554

[27] Matsakis, Nicholas E.: *Recognition of Handwritten Mathematical Expressions*, MIT, Diplomarbeit, 1999. http://www.ai.mit.edu/projects/natural-log/papers/matsakis-MEng-99.pdf

[28] Matsakis, Nicholas: *Natural Log*. http://www.ai.mit.edu/projects/natural-log/demo/. Version: 1999, Abruf: 9-9-2010

[29] Meyer, A: Pen computing: a technology overview and a vision. In: *ACM SIgCHI Bulletin* 27 (1995), Nr. 3, 46–90. http://portal.acm.org/citation.cfm?id=221308

[30] Micheli-Tzanakou, Evangelia: *Supervised and unsupervised pattern recognition*. CRC Press, 2000

[31] Pakin, S: *The comprehensive latex symbol list*. http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf. Version: 2010

[32] Plamondon, R. ; Srihari, S.N.: On-line and off-line handwriting recognition: A comprehensive survey. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* Bd. 22, IEEE, 2000, S. 63–84

[33] Platt, J.C. ; Cristianini, N. ; Shawe-Taylor, J.: Large margin DAGs for multiclass classification. In: *Advances in neural information processing systems* 12 (2000), Nr. 3, 547–553. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.6283&rep=rep1&type=pdf

[34] Rabiner, L. ; Juang, B.H.: *Fundamentals of speech recognition*. Prentice hall Englewood Cliffs, New Jersey, 1993 http://www.citeulike.org/user/asterix77/article/308923

[35] Rabiner, LR: A tutorial on hidden Markov models and selected applications inspeech recognition. In: *Proceedings of the IEEE* 77 (1989), Nr. 2, S. 257–286

[36] Ratanamahatana, C ; Keogh, E: Everything you know about dynamic time warping is wrong. In: *Third Workshop on Mining Temporal and Sequential Data*, 2004

[37] Schölkopf, Bernard ; Smola, Alexander J.: *Learning with Kernels*. The MIT Press, 2002

[38] Smithies, S ; Novins, K ; Arvo, J: A handwriting-based equation editor. In: *Graphics Interface*, 1999, 84–91

[39] Suzuki, M. ; Tamari, F. ; Fukuda, R. ; Uchida, S. ; Kanahori, T.: INFTY: an integrated OCR system for mathematical documents. In: *Proceedings of the 2003 ACM symposium on Document engineering*, 2003, 104

[40] Tapia, E ; Rojas, R: Recognition of on-line handwritten mathematical formulas in the e-chalk system. In: *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on* (2003). http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1227805

[41] Tapia, E ; Rojas, R: Recognition of on-line handwritten mathematical expressions in the e-chalk system-an extension. In: *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on* (2005). http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1575734

[42] Tapia, E ; Rojas, R: A Survey on Recognition of On-Line Handwritten Mathematical Notation. In: *Citeseer* (2007), Jan. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.7996&rep=rep1&type=pdf

[43] Tapia, Ernesto: *JMathNotes*. http://page.mi.fu-berlin.de/tapia/projects.php. Version: 2004, Abruf: 9-8-2010

[44] Tappert, C.C.: Cursive script recognition by elastic matching. In: *IBM Journal of Research and development* 26 (1982), Jan, Nr. 6, 765–771. http://portal.acm.org/citation.cfm?id=1664966.1664979

[45] Tappert, C.C. ; Suen, C.Y. ; Wakahara, T.: The state of the art in online handwriting recognition. In: *IEEE transactions on pattern analysis and machine intelligence* (1990), S. 787–808

[46] Tilkov, Stefan ; Tilly, Marcel ; Wilms, Hartmut: Lose Kopplung mit Web-Services einfach gemacht. In: *Javaspektrum* 5 (2005). http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/js/2005/05/tilly_JS_05_05.pdf

[47] Vuong, B ; He, Y ; Hui, S: Towards a web-based progressive handwriting recognition environment for mathematical problem solving. In: *Expert systems with Applications* 37 (2010), Nr. 1, 886–893. http://linkinghub.elsevier.com/retrieve/pii/S0957417409005193

[48] W3C: *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. http://www.w3.org/TR/SVG/

[49] W3C: *XMLHttpRequest*. http://www.w3.org/TR/XMLHttpRequest/

[50] Watt, SM ; Xie, X: Recognition for large sets of handwritten mathematical symbols. In: *IEEE International Conference on Document Analysis and Recognition (ICDAR)* (2005), S. 39–43

[51] Winkler, H: HMM-based handwritten symbol recognition using on-line and off-line features. In: *Atlanta* 6 (1996), Jan, 3438–3441. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.2454&rep=rep1&type=pdf

[52] Xie, Xiaofang: *On the recognition of handwritten mathematical symbols*, University of Western Ontario, Diss., 2007

[53] XThink: *MathJournal*. http://www.xthink.com. Version: 2010, Abruf: 9-9-2010

[54] Zanibbi, Richard: *FFES*. http://research.cs.queensu.ca/drl/ffes/. Version: 2004, Abruf: 9-9-2010

Abbildungsverzeichnis

Algorithmenverzeichnis