Erkenner von Unicode-Emojis und ASCII-Emoticons sowie entsprechender Neubildungen

Christopher Michael Chandler
Lüner Höhe 7
59174 Kamen
christopher.chandler@rub.de

Gianluca Cultraro
Widumestraße 5
44787 Bochum
gianluca.cultraro@rub.de

Abstract

Seit ihrer Einführung im 20. Jahrhundert sind Emoticons und Emojis mittlerweile ein fester Bestandteil der informellen digitalen Schriftsprache geworden. Aufgrund dessen beschlossen wir zu erforschen, inwieweit sie in der Schriftsprache vertreten sind und ob Personen dazu neigen, bei dem Erstellen und Einsetzen von Emoticons besonders kreativ zu sein. Ferner wollten wir wissen, ob Emojis oder Emoticons stärker vertreten sind. Wir konzentrierten uns dabei hauptsächlich auf vier Kategorien; Seitliche Emoticons in beiden Richtungen, Emojis und Neubildungen, die sich aus ASCII-Zeichen zusammensetzen. Um der Sache auf den Grund zu gehen, verfassten wir anhand Python 3.8 mit dem RE-Modul Emoticonalgorithmen. Es stellte sich heraus, dass trotz der wachsenden Beliebtheit der Emojis diese im Vergleich zu den Emoticons weniger vorkamen. Dazu wurde uns im Laufe unserer Forschung klar, dass Personen bei der Bildung von Emoticons sich sehr an „:)“ orientieren. Dies führte letztendlich dazu, dass sämtliche Neubildungen sich stark ähnelten.

1 Einführung

Diese Dokumentation beinhaltet alle wichtigen Informationen über die Entwicklung und Funktionsweise unseres Emoticons und Emoji-Erkenners. Im Laufe dieser Arbeit wird zwischen Emojis und Emoticons unterschieden, bei deren Definition wir uns an der im Duden orientieren. Als Emoticons definieren wir einzelne Zeichen oder Reihenfolgen von ASCII-Zeichen und beschränken uns auf klassische westliche Emoticons, die seitwärts gelesen werden wie z.B. „:-)“.

Wir konzentrieren uns auf Emoticons aus dem deutsch- und englischsprachigen Raum und verzichten deshalb auf Emoticons, die vertikal gelesen werden und teilweise Symbole anderer Sprachen enthalten wie beispielsweise koreanische Schriftzeichen „ㅠ_ㅠ“ oder die aus dem Japanischen stammenden Kaomoji „(^-^)“. Bei Emojis handelt es sich um Piktogramme, die alle über ihren eigenen und individuellen Unicode-Code definiert sind und so plattformunabhängig und sprachunabhängig sind.

1.1 Herkunft der Emoticons

Dass der Mensch Bildhaftigkeit bei der Implementation von Sprache gerne einsetzt, ist an sich kein neues Phänomen. Wenn man einen Blick auf logografische Sprachen wie z.B. Mandarin, Japanisch und sogar Altägyptisch wirft, stellt man fest, wie wichtig, oder zumindest nützlich, diese Bildhaftigkeit in der Sprache ist. Durch das Aufkommen des digitalen Zeitalters hat sich die Kommunikation unter den Menschen jedoch weitgehend gewandelt; von der klassischen Kommunikation von Angesicht zu Angesicht oder Sprache zu Sprache zur heutigen Schrift zu Schrift Kommunikation.

Das größte Problem bei dieser Kommunikationsart ist die fehlende Möglichkeit visuelle und nonverbale Signale zu senden. Diese Hinweise werden in der textlichen Kommunikation mittlerweile durch Emojis und Emoticons ersetzt. Als Erfinder des Emoticons wird ein Professor im Jahre 1982 betitelt, der einen Kommentar auf dem Bulletin Board eines Institutes für Computerwissenschaften hinterließ. Der Autor erklärte hier die Lesart und Nutzung des ersten Emoticons: „:-)“.

Seit diesem Auftauchen des Emoticons und neuer Kommunikationsplattformen wandelte sich das Aussehen der Emoticons und Emojis zu einer immer bunteren und komplexeren Vielfalt. Die heutige Generation der „Digital Natives“ sehen die Nutzung und den Inhalt von Emoticons und Emojis mittlerweile als selbstverständlich an. Ihre Interpretation hängt jedoch von Generation und Kultur des Lesers ab (Lee, 2016).

1.2 Forschungsfrage

Die Hauptthematik unserer Arbeit ist ein Emoticon- und Emoji-Erkenner. Die Emoticons und Emojis sollen, sowie zugehörige Neubildungen, erkannt und entsprechend kategorisiert werden. Diese Neubildungen sollen von dem Programm gezählt werden, um so später einen möglichen Anhaltspunkt über die Kreativität der Nutzer geben zu können. Das Programm soll in der Lage sein, schon vorgefertigte Textdateien zu verarbeiten sowie

dynamisch auf die Social Media Plattform Twitter zugreifen zu können, um dort öffentlich gepostete Tweets zu untersuchen. Dieser dynamische Zugriff ermöglicht es dem Programm, ohne vorherige Anfertigung eines Textes, an höchst aktuelle Daten für die weitere Verarbeitung zu kommen.

Die Erkennung von Emojis stellt kein Problem dar, da diese über ihren Unicode-Code definiert sind, deren Liste öffentlich zu Verfügung steht. Die Problematik des Erkenners liegt in der Erkennung von Emoticons, da aufgrund der schieren Masse an Kombinationsmöglichkeiten aller ASCII-Zeichen die Vielfalt der möglichen Emoticons unendlich scheint, gerade wenn man die Mehrfachnutzung von Zeichen miteinbezieht. Unter Einbezug einer Liste allgemein bekannter Emoticons kann das Programm entlastet werden, aber dies bietet keine Lösung.

2 Stand der Forschung

Im Folgenden werden relevante Arbeiten genannt, die sich im gleichen Untersuchungsfeld zuordnen lassen, der Untersuchung von Emoticons und Emojis. Bei dem Social Media Web Tagger (SoMeWeTa) handelt es sich um einen Tagger für deutsche Social Media und Web Texte. Das Besondere an diesem Programm ist sein Prozess zur Domänen Adaption (Proisl, 2018). Bei der SemEval 2018 nahmen mehrere Teams teil, um ein Programm zu schaffen, das, gegeben den Text eines Tweets, den Emoji vorhersagen sollte, der am wahrscheinlichsten in dieser Kombination genutzt werden würde.

Bei den meisten Programmen handelte es sich um neuronale Ansätze, die meisten hierbei Label-Wise Attention Mechanismen (LSTM) (Barbieri et al., 2018). Auch Barbieri et al. (2018) kamen zu dem Ergebnis, dass diese LSTMs sich auch gut für Emojis eignen, die nicht oft in den Trainingsdaten vertreten sind, wodurch das Programm unvoreingenommener ist gegenüber höher frequentierten Emojis. Außerdem ermöglicht dieser Ansatz einen gewissen Grad der Interpretation über die Nutzungsweise verschiedener Vorhersage Features (Barbieri et al.,2018).

Ein weiterer Ansatz, der sich auf Tweets von Twitter bezieht, aus dem Feld der Emoji Untersuchung ist die Ersetzung der Piktogramme mit ihrer definierten Beschreibung in normaler Sprache und sie dementsprechend wie Standard Wörter zu behandeln. Dies steigert die Verlässlichkeit von Ironie Erfassung und verbessert die Analyse der Emotionen des Tweets (Singh et al.,2019).

3 Beschreibung des Vorgehens

Das Programm beansprucht mehrere Funktionen und Module, die sowohl von uns definiert wurden als auch in Python 3.8 enthalten sind. Der Übersichtlichkeit und Kürze halber werden wir lediglich auf die wesentlichsten Module und Funktionen näher eingehen. Genauere Erklärungen der Funktionen und Module, die hier nicht erwähnt werden, befinden sich im Code und in der ReadMe-Datei.

Tweepy ist ein notwendiges Modul, womit Tweets von Twitter entnommen werden können. Tweepy muss jedoch dafür via *pip install tweepy* installiert werden. Obwohl Tweepy an sich ein wichtiger Teil des Programms ist, ist es jedoch nicht das wichtigste Modul.

Das wichtigste Python Modul ist RE. Es ist ein Kernbestandteil des Programms, da das Programm sich auf reguläre Ausdrücke regelmäßig beziehen muss. Anhand dessen wird aus der Emoticondatenbank einen Emoticonalgorithmus erstellt. Wir gehen später näher darauf ein, wie genau wir das Modul nutzen.

In erster Linie geht es hauptsächlich darum, Textdateien, als .txt abgespeichert, einzulesen und sie dann anschließend anhand der von uns bestimmten Parametern zu analysieren. Es gibt zwei Möglichkeiten, Dateien in das Programm zu importieren: file_import() für Textdateien und twitter_import() für Tweets gespeichert in einer Textdatei. Die Differenzierung der Funktionen ist hier vor allem für die Benennung der Ergebnisdateien am Ende und den Aufruf von Tweepy wichtig.

Nach dem Einlesen wird der Text wie folgt formatiert, geparst und dann tokenisiert. Es werden einige personenbezogene Daten anonymisiert, damit keine Rückschlüsse auf einzelne Personen gemacht werden können. Dazu gehören beispielsweise E-Mail-Adressen, Hashtags, Webseiten, Benutzernamen, etc.

Es wird in sehr vielen Fällen davon ausgegangen, dass ein Leerzeichen als Token-Grenze dienen soll. Das heißt, dass der Ausdruck “Ein Haus” zwei Tokens enthält. Eine naive Vorgehensweise reicht hier dennoch nicht aus, da wir sinnvolle linguistische Einheiten für eine korrekte Analyse benötigen. Unter sinnvoll verstehen wir alle Tokens, die schon lexikalisiert wurden, sodass sie nicht weiter zerlegt werden können. “Haus” ist eine in sich geschlossene

Einheit, wobei “Haus.” oder “Haus:)” aus semantischen und syntaktischen Gründen nicht sinnvoll sind. Aufgrund dessen wurden solche Vorkommnisse getrennt, sodass am Ende “Haus .” und “Haus :)” entstanden. Danach wurden sie wieder am Leerzeichen getrennt.

Nach dem Tokenisieren werden die Tokens an emoticon_emoji_erkenner weitergegeben. Innerhalb dieser Funktionen finden mehrere Prozesse statt, die zur tatsächlichen Bestimmung der Emoticons beitragen. Inmitten der Funktion wird eine Funktion namens token_filter aufgerufen, die den emoticon_emoji_erkenner vor falschen Ergebnissen schützen soll. Diese Fehler treten vor allem auf, da Emoticons und Wörter Überschneidungen in Bezug auf ihre Komponenten haben. Das heißt, dass “Haus:” als Emoticon falsch eingestuft werden kann, da “s:” bzw. “:s” in dem Token vorkommt. Da dieses ein Wort und kein Emoticon ist, wird es vom Emoticon-Tagging ausgeschlossen.

Das Programm wird mit einem Emoticonkorpus mitgeliefert, worauf sich das der emoticon_emoji_erkenner bezieht und woraus der Algorithmus erstellt wird. Wir definierten ein Emoticon wie folgt:

Ein Emoticon muss mindestens aus zwei Zeichen bestehen, jedoch wurde eine maximale Länge von uns nicht definiert. Das heißt, ein Emoticon könnte zwar theoretisch unendlich sein, aber in der Praxis kam so ein Fall nicht vor.

Basierend auf unserem Grundkorpus stellten wir folgende Komponenten zusammen.

Head	#%&*+,-038:;<=>BDOPX[\dopx{|}
Body	#$&'()*,-./03:;<=>@DLOSX[\]^_bcops{|}~×þ-
Tail	#&()*,./0378:<>?@DEFIJOPQVX[\]bclop|}Þþ

Tabelle 1: Zeichenklassen der regulären Ausdrücke

Head bildet den Kopf des Emoticons. Diese sind z.B. die Augen “:”. Alle Zeichen, die nach dem Kopf, aber vor dem Tail folgen, bilden den Body bzw. den Körper des Emoticons. Diese sind z.B. die Nase “-” oder der Mund “). Somit kämen solche Emoticons wie “:)” “:-” “:-)” zustande. Der Tail bzw. Schwanz des Emoticons ist immer das letzte Zeichen, weswegen es Überschneidungen mit dem Body geben kann. Ein Schwanz muss nicht vorhanden sein, jedoch ein Body. Das heißt, dass “:)” oder “:-)” vorkommen können.

Anhand der Zeichen in der Tabelle werden dann Neubildungen erkannt, wobei Emoticons, die in dem Korpus vorhanden sind, anhand eines Lookups gemacht werden. Dies führt zu einem zuverlässigen Programm, da das Programm nicht komplett auf den Algorithmus angewiesen ist und mit jedem Training wachsen kann.

Bei der Zuteilung der Tags gibt es vier Kategorien: Emoticons, Emojis, Neubildungen und Rest. Emoticons sind alle Zeichen, die aus mindestens einem Head und einem Body bestehen; Tails sind dabei optional. Emojis sind alle Unicode-Emojis, die von dem Unicode Consortium festgelegt wurden (vgl. Unicode Consortium, 2020).

Neubildungen beziehen sich ausschließlich auf die Komponenten der Emoticons. Die letzte Kategorie ist Rest, wozu alle Tokens gehören, die nicht zu den anderen Kategorien zugeordnet werden können.[1] Am Ende der Auswertung gibt es die Möglichkeit, die Ergebnisse im IDLE-Fenster ausgeben zu lassen, als Datei abzuspeichern oder beides.

Unsere Korpora setzten sich hauptsächlich aus zwei Quellen zusammen: Von dem Dortmunder Chat-Korpus wurden zwei Texte entnommen und von Twitter erzeugten wir drei Sorten von Tweetdateien: Englisch, Deutsch und Global.

Für die englischen Tweets wurde nach allen Tweets gesucht, die ":D“ und „and" enthielten. Für die Deutschen Tweets mit „:D“ und „und" und die globalen Tweets mussten „:D“ „(:“ enthalten. So erstellten wir ein Training- und Testkorpus für unser Programm. Alle drei wurden mithilfe eines Internet-Applets namens IFTTT erzeugt (vgl. IFTT, n.d.).

4 Evaluation

Die Auswertungen der Dateien lief in zwei Schritten ab; Trainingsphase und Testphase. Die Trainingsphase diente dazu, die Algorithmen des Erkenners und des Filters zu trainieren und zu präzisieren. Anschließend folgte die Testphase, wo die tatsächliche Zuverlässigkeit des Programms auf die Probe gestellt wurde.

[1] Dieser Tag kommt am häufigsten vor. Die Gründe dafür werden in der Evaluation näher erläutert.

Der Aufbau sowie der Verlauf beider Phasen waren identisch. Es wurden vier Texte pro Phase, die sich in Anzahl an Tokens und Länge ähnelten, ausgesucht: 100 Zeilen Tweets, jeweils Englisch und Deutsch. Eine globale Tweetdatei, die sich aus mehreren Sprachen, teilweise aus nicht-europäischen Sprachen, zusammensetzte. Die letzte Datei entstammte aus dem Dortmund-Chatkorpus.

Ein interessantes Ausgangsproblem beim Taggen war die Tatsache, dass lexikalische Wörter mit einer ziemlich hohen Trefferquote fälschlicherweise als Emoticons erkannt wurden, da Wörter und Emoticons häufig die gleichen ASCII-Zeichen verwenden. Aufgrund dessen war es nicht möglich, einen naiven Ansatz zu verwenden z.B. Wenn ":" in einem Token vorkommt, ist es ein Emoticon. Dieses funktioniert nicht, da man zwischen ":D" und "Punkte:" unterscheiden muss.

Die Tabellen der Ergebnisse sind wie folgt zu deuten:

	Bedeutung
En	Englische Tweets
De	Deutsche Tweets
Global	Globale Tweets
Dortmund	Dortmunder Chat Corpus
:)	Emoticon
😃	Emoji
:-)	Neubildung
Rest	Sonstige Tokens
%	Fehlerrate

Tabelle 2: Legende der Ergebnisse

Bei der Trainingsphase tauchten einige Problemfälle auf. Emoticon und Emojis als solche gehörten mehr oder minder zu einem festen Lookup. Daher war es nicht problematisch, sie entsprechend zu differenzieren bzw. zu erkennen.

Für den Erkenner war es jedoch nicht immer ganz klar, was als Neubildung und was als Rest gelten soll. Dies lässt sich anhand von einigen Beispielen veranschaulichen; *<3 &amp*; und *>_>*

Bei allen drei sieht man, dass sie die Symbole '; ', '&' teilen. Diese drei Symbole tauchen häufig bei den Emoticons auf z.B. bei "**;-(**" und "&-|**"**. Diese führten die Algorithmen zu falschen Ergebnissen. Um diese Problematik zu beheben, wurden die Algorithmen angepasst, sodass die Kontrollen und Bestimmungen stärker ausfielen. Des Weiteren wurde "amp" und sämtliche Variationen im Zusammenhang mit Zeichen z.B. 'amp;' ausgeschlossen.

Die Ergebnisse der Dortmunder Texte halfen zwar beim Trainieren des Programms, jedoch sind die Bewertungen der Ergebnisse problematisch, da die Technische Universität Dortmund ihre eigenen Golddateien als .xml Format festgelegt hatten wie z.B. <emoticon>:) </emoticon>.

Dies ließ sich nicht mit unserem Programm vereinbaren, da unsere Golddateien ein Spaltenformat aufweisen.

Emoticon	**Programmtag**	**Goldtag**
:)	Emoticon	Emoticon

Tabelle 3: Ergebnisformat

Nichtsdestotrotz benutzen wir ihre Golddatei und unsere, um die Zuverlässigkeit des Programms zu bestimmen. Zudem waren die Dortmunder Texte beim Trainieren und Gestalten des Programms ausschlaggebend, da sie relativ lang und vielfältig in Bezug auf Sprachlichkeit ausfallen.

4.1 Trainingsphase

	En	De	Global	Dortmund
:)	101	112	2025	35
😃	81	2	335	N.A. [2]
:-)	13	2	57	1
Rest	3409	2648	29650	18965
%	0.31	0.07	0.08	0.02 -0.159

Tabelle 4: Bewertungsübersicht der Trainingsphase

Bei allen Texten sieht man, dass der Tag Rest überwiegend präsent ist im Vergleich zu den anderen Tokens. Wenn man sich Emoticons und Emojis anschaut, stellt man relativ schnell fest, dass Emoticons und Neubildungen im Vergleich zu den Emojis viel häufiger vorkamen. Neubildungen an sich waren selten.

Diese Problematik war bei der Erstellung der Golddatei besonders wichtig, da nicht immer zu erkennen war, ob der Verfasser des Tweets sich vertippte oder ob wirklich eine Neubildung vorlag. Kombinationen, wie z.B. :D" :)#V und #X1 kamen häufig vor und wurden deswegen als Neubildungen eingestuft.

4.2 Testphase

	En	De	Global	Dortmund
:)	105	104	2006	235
😃	7	81	671	N.A.
:-)	1	2	25	20
Rest	2892	3297	41713	26923
%	0	0.06	0.01	0.01 - 1.03

Tabelle 5: Gesamtbewertung der Testphase

[2] N.A. (Eng. not applicable, Deu. nicht anwendbar). Es kommen in diesen Texten keine Emojis vor. Somit werden sie bzw. können nicht gezählt werden.

Die Testphase fiel ähnlich aus, wobei die möglichen Neubildungen weniger präsent sind als bei den Testdaten, obwohl der Datensatz bzw. die Anzahl an Tokens größer waren. Das hing vor allem damit zusammen, dass das Korpus soweit trainiert wurde, dass es für den Erkenner weniger Tokens gab, die als Neubildungen eingestuft werden konnten. Sie wurden nicht mehr als neu erkannt, sondern als Emoticons, die sich im Korpus befanden.

5 Diskussion der Ergebnisse

Die Ergebnisse fielen wie erwartet aus. Es war stark davon auszugehen, dass ein Großteil des Textes nicht aus Emojis, Emoticons und deren Neubildungen bestehen konnte. Es wäre möglich gewesen, Emoticon und Emoji-lastige Texte als Datensätze zu nehmen. Diese hätte jedoch aus den folgenden Gründen die Realität weniger abgebildet.

Emoticons, Neubildungen und Emojis erscheinen hauptsächlich eher in informellen Kontexten wie z.B. in Chatgesprächen und Forumsbeiträgen. Wenn sie allerdings erscheinen, dann sporadisch und nicht in der Überzahl, denn ein Gespräch, zumindest im üblichen Sinn, kann nicht aus reinen Nichtwörtern bestehen. Aufgrund dessen kann sie als eine Ergänzung und nicht als Ersatz des Gesprächs angesehen werden.

Es ist schwierig an solche Gespräche zu kommen, ohne dass man die Texte künstlich gestaltet. Dies würden wiederum die Ergebnisse bzw. Zuverlässigkeit des Programms verfälschen.

Die folgenden Emojis, Emoticons und Neubildungen, ohne eine Aussage über deren Deutung und deren Einfluss auf den Text zu liefern, sind wie folgt:

	Emojis	Emoticon	Neubildung
#1	❤️ (45)	:) (3035)	:)))))))))))) (6)
#2	😂(45)	:D (600)	:)) (5)
#3	✨(41)	:)) (363)	;-))) (4)

Tabelle 6: Ergebnisse der häufigsten Tokens

Was die Neubildungen betrifft, sieht man, dass sie einigermaßen einheitlich in Bezug auf deren Form bzw. Variationen von ":)" sind.

Abschließend sollte erwähnt werden, dass die Erkennung der Neubildungen zwischen den Testphasen und Trainingsphasen anders ausfielen, da der Erkenner mit jedem Text die Möglichkeit hatte, das Korpus auszubauen und Neubildungen als Emoticons einzustufen. Aus diesem Grunde sank die Anzahl der Neubildungen. Würde man den Erkenner mit einem kleineren Korpus ausführen, würde die Anzahl der Neubildungen dementsprechend steigen.

6 Fazit

Als Anhaltspunkt und Inspiration zur Erstellung unseres Erkenners orientierten wir uns an mehreren Programmen, wie zum Beispiel STACCADo aus Dortmund oder den SoMeWeTa-Tagger.

Im Vergleich zu den anderen Programmen bestand die Aufgabe unseres Programms nicht darin, irgendwelche semantischen Aussagen über den Text zu treffen oder anhand schon tokenisierter Texte, Ergebnisse zu liefern; es ging vielmehr darum, festzustellen, was genau als Emoticon gelten soll und wie zugehörige Neubildungen aussehen müssen bzw. können. Hinsichtlich dieser Annahme wurde uns im Laufe der Erstellung des Programms klar einiges.

Wir gingen anfangs davon aus, dass Emojis in der Überzahl wären und dass Emoticons nicht in einer erwähnenswerten Menge vorkämen. Beide Aussagen stellten sich jedoch als falsch heraus.

Obwohl Emojis selbstverständlich in den Texten vorkommen, waren es deutlich weniger als anfangs angenommen. Dafür kann es mehrere Gründe geben, wie z.B. die Korpora, die Zusammensetzung der Korpora, etc. Eine andere Erklärung, die unserer Meinung nach viel plausibler, ist, ist dass es nicht immer möglich ist, Emojis zu erzeugen, vor allem wenn man die Texte am Rechner und nicht am Handy erzeugt.

Unserer Ansicht nach stellt dies keine Problematik dar, denn die Texte spiegelten die Realität wider. Die Texte stammen aus mehreren Ländern und verschiedenen Sprachen und somit kann gewährleistet werden, dass keine künstlichen Texte und Quellen vorliegen, die die Ergebnisse verfälschen konnten.

Das Programm erfüllte unsere Erwartungen und erwies sich als sinnvoll. Die Fehlerrate des Programms war niedrig und die richtige Zuordnung des Tags funktioniert wie erwünscht. Es war auch möglich, dem Programm mehrere unbekannte Texte zu liefern und ein sinnvolles Ergebnis zu bekommen.

Nichtsdestotrotz ist uns bewusst, dass Verbesserungen möglich sind. Es wäre denkbar, dass das Programm Emoticons und Neubildungen in Kategorien einteilen könnte, zum Beispiel Smiley, Sad, Love, etc. um somit eine Aussage über die Gefühlslage des Textes unter Vorbehalt einer absoluten Deutung zu ermöglichen.

Außerdem wäre es möglich, dass Programm in Bezug auf Emojis flexibler zu gestalten. Da das Programm sich auf ein Emojikorpus verlassen muss, das nicht mit der Zeit wachsen kann, ist es nicht möglich, neuere Emojis zu erkennen. Dies lässt sich anhand komplexer regulärer Ausdrücke lösen.

Es muss jedoch gesagt werden, dass wir uns bewusst dagegen entschieden, denn das Programm kann so umgeschrieben werden, sodass die Kürzel der Emoticons bzw. CLDR Short Name mit ausgegeben werden können.

Obwohl die genannten Punkte in unserem Programm fehlen und das Programm mit Sicherheit verbessert hätten, wären solche Änderungen entweder nicht möglich oder ziemlich umständlich gewesen. Sie hätten auch nicht unbedingt zu einer funktionalen Optimierung des Programms geführt. Somit sind wir zuversichtlich, dass das Programm trotzdem sinnvoll und zuverlässig ist.

Literaturverzeichnis

Barbieri, F., Camacho-Collados, J., Ronzano, F., Espinosa-Anke, L., Ballesteros, M., Basile, V., Patti, V., Saggion, H. (2018). SemEval 2018 Task 2: Multilingual Emoji Prediction. In *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)* (pp. 24–33). New Orleans, Louisiana: Association for Computational Linguistics. doi: 10.18653/v1/S18-1003

Barbieri, F., Espinosa-Anke, L., Camacho-Collados, J., Schockaert, S., Saggion, H. (2018). Interpretable Emoji Prediction via Label-Wise Attention LSTMs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 4766-4771). Brussels, Belgium: Association for Computational Linguistics. doi: 10.18653/v1/D18-1508

Ifttt. (n.d.). *Applets*. https://platform.ifttt.com/docs/applets

Lee, J. S. (2016). Emoticons. In S. Moran (Ed.), *Ethical Ripples of Creativity and Innovation* (pp. 207-213). Palgrave Macmillan.

List of emoticons. (2020, July 12). In *Wikipedia*. https://en.wikipedia.org/wiki/List_of_emoticons

Proisl, T. (2018). SoMeWeTa: A Part-of-Speech Tagger for German Social Media and Web Texts. In Calzolari N, Choukri K, Cieri C, Declerck T, Goggi S, Hasida K, Isahara H, Maegaard B, Mariani J, Mazo H, Moreno A, Odijk J, Piperidis S, Tokunaga T (Eds.), In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)* (pp. 665–670). Miyazaki, JP: Miyazaki: European Language Resources Association.

Singh, A., Blanco, E., Jin, W. (2019). Incorporating Emoji Descriptions Improves Tweet Classification. In *Proceedings of NAACL-HLT 2019* (pp. 2096-2101). Minneapolis, Minnesota: Association for Computational Linguistics. doi: 10.18653/v1/N19-1214

Technische Universität Dortmund. (2006). *Releasekorpus : HTML-Version*. https://www.uni-due.de/germanistik/chatkorpus/

Tweepy. (n.d.). *Tweepy Documentation*. http://docs.tweepy.org/en/latest/

Twitter. (2013). *GET statuses/ sample — Twitter Developers*. https://dev.twitter.com/docs/api/1/get/statuses/sample.

Unicode Consortium. (29, Januar 2020). *Full Emoji List,v13.0.* https://unicode.org/emoji/charts/full-emoji-list.html

YouSmiley. (n.d.). *ASCII Smileys*. http://www.yousmiley.de/ASCII-Smileys.html