

Multi-Class Detection of Abusive Language Using Automated Machine Learning

Mackenzie Jorgensen¹, Minhho Choi², Marco Niemann³, Jens Brunk³, and Jörg Becker³

¹ Villanova University, Dept. of Computing Sciences, Villanova, USA

² Lewis & Clark College, Dept. of Mathematical Sciences, Portland, USA

³ University of Münster – ERCIS, Münster, Germany

mjorgen1@villanova.edu, minhochoi@lclark.edu,

marco.niemann@ercis.uni-muenster.de,

jens.brunk@ercis.uni-muenster.de, becker@ercis.uni-muenster.de

Abstract. Abusive language detection online is a daunting task for moderators. We propose Automated Machine Learning (Auto-ML) to semi-automate abusive language detection and to assist moderators. In this paper, we show that multi-class classification powered by Auto-ML is successful in detecting abusive language in English and German as well as and better than the state-of-the-art machine learning models. We also highlight how we combatted the imbalanced data problem in our data-sets through feature selection and undersampling methods. We propose Auto-ML as a promising approach to the field of abusive language detection, especially for small companies who may have little machine learning knowledge and computing resources.

Keywords: Abusive Language Detection, Automated-Machine Learning, Multi-Class Classification

1 Introduction

Online users post approximately 500M tweets a day [1]. Between August 2015 and July 2016, the Anti-Defamation League found 2.6M tweets containing anti-Semitic language and about 20,000 of those tweets targeted 50,000 journalists [2]. An Amnesty International study revealed that 23% of women across eight countries had experienced online abuse or harassment [3]. The evidence for the abusive language problem towards journalists and minority groups on social media is overwhelming. Early detection of abusive language is crucial to the safety of these online spaces and for news media's regulatory compliance.

Social media companies, who lack an end-to-end abusive language detection system, and small and medium-sized companies, who lack the resources to check for problematic comments manually, need a solution for abusive comment and post moderation. A promising approach is semi-automation through Machine Learning (ML), which has shown success in abusive language detection [4, 5]. One example is

the Coral Project's platform Talk¹ which includes comment moderation tools powered by ML and Jigsaw's Perspective API² (providing a toxicity score given a piece of speech) for moderators.

Researchers usually choose what decisions to make along the ML pipeline (a tedious and time-consuming process) before obtaining a model that provides decent results depending on the hardware used and the project's scale. Automated Machine Learning (Auto-ML) provides an approach to potentially alleviate the time-consuming ML pipeline decisions from researchers. It helps researchers test a multitude of parameters and methods at once, not solely focusing on the optimization of a single ML algorithm. Non-ML experts can easily use Auto-ML, even with having little to no knowledge of the workings in the background [6]. To the best of our knowledge, this pathway has so far remained unexplored.

The literature to date on abusive language detection overwhelmingly focuses on English data-sets [4, 7, 8]. However, abusive language occurs in all languages. Thus, we included English and German abusive language data-sets in our research to begin addressing the lack of multilingual abusive language detection research. German has many complexities that English does not, making German abusive language detection rather challenging [5]. We utilize the term "abusive language" throughout our paper as an umbrella term which includes concepts like hate and offensive speech [7]. In recognizing that abusive language, most of the time, is non-binary and often has many different intricacies and types, we utilize data-sets which include multiple classes to account for the sub-concepts of the abusive language umbrella online [4, 5].

Our paper is organized as follows: Section 2 contains a literature review of the state of the art research, Section 3 contains an explanation of our experimental design including insights into our data-sets and our methods for preparing the data and testing with *H2O Auto-ML*, Section 4 contains a summary of our results, and Section 5 contains a discussion of the impacts of our work and future research.

2 State of the Art

2.1 Abusive Language Detection

For English abusive language detection, we mainly focused on hate speech detection. Although there are diverse definitions of hate speech, we focused on the detection process of hate speech, and thus, we will not cover descriptions of hate speech in detail. We explored various data-sets available for hate speech detection. Waseem and Hovy, Chatzakou et al., and Davidson et al., compiled the three most commonly used English Twitter data-sets in the hate speech detection domain [4, 8, 9]. The researchers categorized tweets using different labels, and they followed distinct procedures for text classification. We describe the three publications' data-sets in-depth in Table 1.

¹ <https://coralproject.net/talk/>

² <https://www.perspectiveapi.com/>

Waseem and Hovy [8] investigated the impact of using different features in multiclass text classification. They employed a logistic regression classifier with 10-fold cross-validation for their model and trained the model using various features. They found that character n-gram and gender information provided a solid foundation for hate speech detection. Chatzakou et al. [9] provided a methodology to detect aggressive and or bullying behavior on Twitter. They mainly focused on tree-based ML algorithms and found that Random Forests resulted in the most promising outcomes. They concluded that bullies post less and are less popular, while aggressors are relatively popular and write more negative posts. Davidson et al. [4] tried to distinguish between hateful and offensive tweets using multi-class ML classifiers. They employed various ML algorithms and found that Logistic Regression and Linear SVM performed better than other models. Their results revealed that racist and homophobic tweets were more frequently classified as hate speech by their model, while it classified sexist tweets as offensive.

There are few labeled German Twitter data-sets available publicly, and Wiegand et al.'s data-set is the only data-set for abusive language detection that we found [5]. They summarized the results of the GermEval 2018 Shared Task on Identification of Offensive Language in German Twitter. In total, 20 teams participated in the task, and each team employed different pre-processing techniques, features extraction methods, and ML classifiers. They concluded that word embeddings, character n-grams, and lexicons of offensive words were vital features and ensemble methods often improved their outcomes.

Table 1. Data-sets Details. For details on previous researchers' data-sets, we include the number of tweets (#T), the language of the tweets (Lang), the labels for the data-sets and the ratios (Lbl), the number of annotators (#A), and the inter-coder agreement rate (Agree.).

	#T	Lang	Labels (Ratio)	#A	Agree.
Waseem and Hovy [8]	16,914	English	racist (0.12), sexist (0.20), normal (0.68)	1	0.84
Chatzakou et al. [9]	9,484	English	aggressive (0.034), bullying (0.045), spam (0.318), normal (0.603)	5	0.54
Davidson et al. [4]	24,802	English	hateful (0.06), offensive (0.77), neither (0.17)	3+	0.92
Wiegand et al. [5]	8,541	German	abuse (0.204), insult (0.119), profanity (0.014), other (0.663)	3	0.66

2.2 Auto-ML

Auto-ML aims to automate the majority of the ML pipeline workflow, including the preprocessing, feature engineering, algorithm selection, and hyperparameter optimization [10, 11]. Auto-ML has the potential to help developers reduce time and costs when running ML algorithms. Further, ML developers prioritize features and algorithms differently, which creates a bias in the ML pipeline process [12]. Auto-ML can alleviate this bias by automating the processes where biases can occur in the first place. Thus, Auto-ML potentially can reduce bias in algorithm selection and assist users in the optimization process to find the optimal algorithms and hyper-parameters.

We identified and evaluated several publicly available Auto-ML frameworks, including *H2O Auto-ML* [13], *auto-sklearn* [14], *Auto-WEKA* [15], and *TPOT* [16]. None of the Auto-ML frameworks listed are compatible with NLP specifically, and some frameworks currently do not support text classification. Since Auto-ML is a developing field, most researchers have applied Auto-ML frameworks to only a few data-sets and problems. One Auto-ML framework that can currently handle NLP is Google's *AutoML Natural Language*.³ However, the source code and a detailed description of how it works are not publicly available. Moreover, companies have to pay Google for access to Google's services for better performance and time efficiency, which is not trivial for small companies. To our knowledge, no literature has delved into how to apply Auto-ML with NLP.

H2O Auto-ML provides a well-documented framework that could be conveniently adapted to our needs. *Auto-sklearn* includes a fair amount of documentation about its framework but uses outdated package versions, which made it incompatible with other up-to-date packages that were necessary for our research. A third framework, *Auto-WEKA*, currently lacks updated information and support about the framework's implementation. Lastly, *TPOT* is a robust and transparent framework, but even though our data-sets were comparatively small, we were unable to obtain competitive results within a reasonable time frame. Hence, we decided to drop the *TPOT* framework as well. Therefore, we used *H2O Auto-ML* for our study, and we compare its performance to that of other ML classifiers tested in previous research.

3 Experimental Design

We tested *H2O Auto-ML* with neither feature selection nor undersampling algorithms as our baseline. We ran our baseline model for 30 minutes, one hour, and two hours. The German baseline included 3,013 features and 7,566 samples in the training data-set, while the English baseline included 11,712 features and 22,304 samples. Due to limited time and resources, we decided to run all feature selection and undersampling runs for 30 minutes. After this testing, we chose the best feature selection and undersampling method combinations to produce optimal results. Once we found the optimal feature selection and undersampling method combination, we tested for different lengths of time to determine if that changed our results. Our training and testing data were split in a 9:1 ratio. For our purposes, we used *H2O Auto-ML* mainly for hyper-parameter optimization and model selection.⁴

The major aim of abusive language detection is to find as many potentially abusive posts as possible. Hence, we focused on recall, since recall highlights how many tweets with abusive language the model detected, while precision tells us how many tweets contained abusive language out of the tweets the model predicted to have abusive language. Moreover, improving recall is beneficial for assisting the moderator, because higher recall means the model will find as many potentially

³ <https://cloud.google.com/natural-language/automl/docs/>

⁴ Our code is accessible here: https://github.com/mjorgen1/DAADRISSE_AbusiveLangProject

hateful and insulting tweets as possible, at which point the moderator can filter through the tweets that the model has predicted as abusive. We include four macro average metrics (metrics computed independently for each class and then averaged): precision (P), recall (R), F-1 (F), and accuracy (A), for comparison. All metrics listed in Tables 3, 4, 5, and 6, are macro average values. When we refer to the metrics in this paper, we assume that they are the macro average values, if not specifically mentioned otherwise.

3.1 Data

We include details about the two imbalanced data-sets for English and German in Table 1. The English dataset we used was originally published by Davidson et al. [4]. One of the main reasons we chose their data-set is because it had the highest inter-coder agreement amongst the four data-sets we compared in Table 1. They included their recall confusion matrix, their results (which we compared to our results), and their framework on GitHub.⁵ Additionally, we utilized Wiegand et al.'s [5] publicly available German data-set.⁶ They labeled a tweet as “profanity” when “profane words are used, however, the tweet does not want to insult anyone.” They also stated that profane words might have been used in “positive sentiment to express emphasis.” Hence, we concluded to merge the “profanity” labels into the “other” labels. The finalized label ratios were as follows: 20.4% as “abuse,” 11.9% as “insult,” and 67.7% as “other.” Although we cannot replicate their results because they do not include a link to their code repository, there are detailed descriptions of the pre-processing, features, results, and machine learning classifiers in their paper.

3.2 Pre-processing

For text pre-processing, we followed the methods used by Davidson et al. [4]. First, we replaced multiple white spaces with one single space, and we removed URLs and mentions in tweets. For German tweets, we replaced umlauts because Python struggles with processing *utf-8* text. Then, we tokenized tweets using the regular expression, *re*, Python package [17]. When tokenizing, we changed tweets to lower-case and stemmed tweets with *Porter stemmer* for English [18] and with *Cistem* for German [19]. For English tweets, we employed a basic tokenizer to tokenize tweets without stemming for POS tagging. Lastly, we utilized stop words from the Natural Language Toolkit, *nltk*, package in Python and included a few other terms that were unnecessary in both English (e.g., “ff” and “rt”) and German (e.g., “lbr”) [20].

3.3 Features

We followed similar feature extraction processes to Davidson et al.'s [4] but made a few alterations. We gathered word unigram, bigram, and trigram features with their

⁵ <https://github.com/t-davidson/hate-speech-and-offensive-language>

⁶ <https://github.com/uds-lsv/GermEval-2018-Data>

TF-IDF weighting through the *TfidfVectorizer* in *sklearn* [21]. We applied part-of speech (POS) tagging to English tweets only, resulting in unigram, bigram, and trigram POS tags from the *TfidfVectorizer* but with different parameters than before. Moreover, we used the Valence Aware Dictionary and Sentiment Reasoner, *VADER* [22] to perform sentiment analysis for English tweets and the *TextBlob* sentiment analyzer [23] for German tweets. For other linguistic features, we gathered the number of syllables, characters, words, and unique terms in tweets and the average number of characters and syllables in each word. We calculated Flesch Kincaid Grade and Flesch Reading Ease scores, revealing the level of writing and readability of each tweet. Lastly, we included the number of URLs, mentions, and hashtags in tweets. The binary response, if a tweet is a retweet, was only included for English.

3.4 Feature Selection

To avoid the curse of dimensionality, which renders itself in high-dimensional data spaces with mainly sparse data, we utilize feature selection techniques [24]. We tested two algorithms for feature selection, where both algorithms are from the *sci-kit learn* package in Python: univariate and feature importance [21]. We utilized these two algorithms, univariate and feature importance, for testing because they provided the most promising results in a comparable time frame. Univariate feature selection works by taking the *k* features with the best scores computed by univariate statistical tests with a classification object (in our case, an *ANOVA F-value* object) [25]. Feature importance finds the top *k* features by computing each feature importance score through a tree-based estimator. For the English and German data-set, we selected 500 and 1,000 features. Since the total number of English features extracted was large, we also selected 3,000 and 5,000 features. We highlight the top 10 univariate and feature importance features selected for English and German in Table 2.

Table 2. We show the top 10 features selected from a univariate feature selection run and a feature importance (FI) run for 1,000 features.

	English		German	
	Univariate	FI	Univariate	FI
1	“bitch”	vader neg	“dumm”	num chars
2	vader neg	“trash”	“strunzdumm”	FRE
3	vader neu	“bitch”	“mosl”	FKGL
4	vader comp.	vader comp.	“schmarotz”	avg syl/word
5	“faggot”	vader neu	“pack”	num terms
6	“trash”	“yanke”	“paedophil”	num syllables
7	“bird”	“bird”	“merkel”	num unique words
8	“yanke”	“yellow”	“invasor”	num words
9	“nigger”	“hoe”	“islam”	num mentions
10	FRE	“charli”	“asyla”	“dumm”

Legend: “” = tfidf features / vader = VADER / FKGL = Flesch Kincaid Grade Level / FRE = Flesch reading Ease

3.5 Resampling

To increase the overall and minority classes' R scores, we introduced an additional resampling step [25], which has been proven to help solve the imbalanced data problem [26]. For oversampling, we used the Synthetic Minority Oversampling Technique, *SMOTE*, to create more samples for our minority classes from the *imblearn* Python package [27, 28]. However, the results were not promising. After further thought, we concluded that creating artificial samples for a text classification problem is not a robust solution to our imbalanced data problem because when we write a synthetic text sample in a comment form, it might not make coherent sense. Thus, we utilized undersampling for the majority classes.

The majority classes included the “offensive” and “neither” classes (for English), and the “other” and “abuse” categories (for German). We tested four methods for undersampling, all included in the *imblearn* Python package: random undersampling, cluster centroids, instance hardness threshold, and condensed nearest neighbors [28]. The random undersampling algorithm is a straightforward method which randomly samples from the majority classes. The cluster centroids algorithm clusters the samples and then takes the centroids from those clusters and replaces the original samples with the centroids. The instance hardness threshold algorithm utilizes a classifier's predictions to remove samples which have a low probability. The condensed nearest neighbor algorithm loops through the samples and iteratively removes samples or keeps them through the 1-nearest neighbor algorithm. We tested the undersampling algorithms with and without feature selection algorithms (univariate and feature importance).

4 Results

4.1 Baseline Results

For our baseline model with neither feature selection nor undersampling methods, we concluded that there was no significant improvement in the performance as the runtime increased. *H2O Auto-ML*'s performance running for 30 minutes was similar to the performances of running for 1 hour and 2 hours. Thus, for the rest of our testing, until we found the optimal feature selection and undersampling pairing, we ran *H2O Auto-ML* for 30 minutes. The 30 minutes baseline run is found in Tables 3, 4, 5, and 6 and the P and A were high for the baseline because the model gave preference to predicting the majority class correctly. The R and F were lower for the baseline because the model suffered from the imbalanced data problem.

4.2 Feature Selection Results

English: We tested each feature selection algorithm for the top 500, 1,000, 3,000, and 5,000 features from the English data-set with *H2O Auto-ML*. We found that a lower number of features resulted in better performance, and the optimal number of features

for English was 1,000. The top two results for English are in Table 3. As shown in Table 3, the univariate feature selection with 1,000 features outperformed the other methods in all four metrics. As expected, this resulted in a high P and A [29]. Hence, the P and A for each class were high, but the R for the "hateful" class was low. For the optimal run combination, we tested 1,000 univariate selected features with undersampling methods to determine if those methods increased the R and F .

Table 3. When only running feature selection algorithms with H2O Auto-ML, these are the macro \mathcal{P} , \mathcal{R} , \mathcal{F} , and \mathcal{A} we found. We compare these results to our baseline run with neither feature selection nor undersampling.

	<i>English</i>				<i>German</i>			
	\mathcal{P}	\mathcal{R}	\mathcal{F}	\mathcal{A}	\mathcal{P}	\mathcal{R}	\mathcal{F}	\mathcal{A}
Baseline	0.76	0.63	0.66	0.88	0.67	0.51	0.54	0.75
Uni. 500	0.79	0.71	0.73	0.90	0.69	0.56	0.60	0.77
Uni. 1,000	0.80	0.71	0.74	0.90	0.70	0.57	0.61	0.77
Imp. 500	0.78	0.69	0.71	0.90	0.66	0.55	0.58	0.76
Imp. 1,000	0.77	0.70	0.71	0.90	0.67	0.54	0.57	0.76

German: We tested each feature selection algorithm for the top 500 and 1,000 features from the German data-set with *H2O Auto-ML*. For both univariate and feature importance runs, the correct classification for the majority class as the “other” class’ R was 0.93 for all four runs. However, the correct classification of the minority classes suffered as the “insult” class’ R ranged from 0.25 and 0.29, and the “abuse” class’ R ranged from 0.44 and 0.51 for the four runs. Running *H2O Auto-ML* with 500 versus 1,000 features did not result in a significant difference as the results in Table 3 reveal; the results were almost all within a hundredth decimal point from one another. The difference between univariate and feature importance was rather small, but univariate had better R and F ; thus, we chose to focus on that algorithm for our optimal combination. For the optimal run combination, we tested for both 500 and 1,000 univariate selected features to determine if undersampling or time made an impact on R and F .

4.3 Undersampling Results

English: The results for our undersampling methods and *H2O Auto-ML* combined are listed in Table 4. We found that random undersampling performed the best compared to the other algorithms with regard to R and F .

Table 4. When only running undersampling algorithms with H2O Auto-ML, these are the macro \mathcal{P} , \mathcal{R} , \mathcal{F} , and \mathcal{A} we found. We compare these results to our baseline run with neither feature selection nor undersampling. (* = run failed)

	<i>English</i>				<i>German</i>			
	\mathcal{P}	\mathcal{R}	\mathcal{F}	\mathcal{A}	\mathcal{P}	\mathcal{R}	\mathcal{F}	\mathcal{A}
Baseline	0.76	0.63	0.66	0.88	0.67	0.51	0.54	0.75
Random	0.68	0.82	0.71	0.82	0.52	0.59	0.53	0.61
Condensed*	X	X	X	X	0.59	0.56	0.57	0.72

Instance	0.64	0.67	0.53	0.53	0.50	0.55	0.44	0.46
Cluster	0.58	0.57	0.43	0.40	0.43	0.44	0.27	0.26

Since the computation of condensed nearest neighbors failed with the full feature space, we concluded to test it in the final combinations since decreasing the matrix size through feature selection could help decrease computation time. As we were mainly focusing on R , we considered random undersampling, which had 0.19 higher R than the baseline R . Therefore, we used random undersampling and condensed nearest neighbor methods with the univariate feature selection algorithm for our final results.

German: The German results for only utilizing undersampling methods with *H2O Auto-ML* are in Table 4. Random undersampling performed the best since it had the highest R and the values for the individual class R support this: “other” as 0.63, “insult” as 0.54, and “abuse” as 0.60. Instance hardness threshold had relatively good minority class R scores (“insult” was 0.63 and “abuse” was 0.63), but the majority class R suffered as “other” was 0.38. Instance hardness thresholds’ R was not as promising as random undersampling’s R , but it was the second-best. Condensed nearest neighbors had the highest F , most likely because of the high R for the “other” class as 0.85; even though the “insult” class’ R was only 0.38 and the “abuse” class’ R was 0.44. The last undersampling method we tested was cluster centroids, and it yielded the worst results of the four methods. The individual class R was 0.14 for “other,” 0.76 for “insult” (relatively high), and 0.42 as “abuse.” Given our search for optimal R , especially for abusive comments, we further focus on random undersampling.

4.4 Optimal Combination Results

English: We highlight the four combinations resulting in competitive outcomes relative to Davidson et al.’s results [4] in Table 5. The four combinations we tested include: univariate feature selection with 500 features plus random undersampling (E1), univariate feature selection with 1,000 features plus random undersampling (E2), univariate feature selection with 500 features plus condensed nearest neighbor (E3), and univariate feature selection with 1,000 features plus condensed nearest neighbor (E4). For all four combinations, our R was equal to or greater than the R of Davidson et al. [4]. The E3 and E4 runs had the same results for all four metrics. The F and A for E3 and E4 are only 0.01 lower than Davidson et al.’s F and A , and P for E3 and E4 are 0.02 lower than Davidson et al.’s P . Hence, we concluded that the performance of E3 and E4 are competitive to that of Davidson et al.’s performance. Moreover, when we examined the R for the minority class with the “hateful” label, we found that E3 was 0.65, E4 was 0.62, and Davidson et al. was 0.61. The E3 run performed better than E4 and Davidson et al. in terms of R for the minority class.

Table 5. Top results from combining feature selection and undersampling methods with H2O Auto-ML for the English data-set for the macro-average \mathcal{P} , \mathcal{R} , \mathcal{F} , and \mathcal{A} . We compared our metrics to the top-performing ML model’s metrics from Davidson et al. [4] and our baseline run’s metrics with neither feature selection nor undersampling.

<i>English</i>	\mathcal{P}	\mathcal{R}	\mathcal{F}	\mathcal{A}
Davidson et al. [4]	0.75	0.81	0.78	0.89
Baseline	0.76	0.63	0.66	0.88
Uni. 500 + Random (E1)	0.71	0.81	0.75	0.87
Uni. 1,000 + Random (E2)	0.70	0.83	0.73	0.83
Uni. 500 + Condensed (E3)	0.73	0.82	0.77	0.88
Uni. 1,000 + Condensed (E4)	0.73	0.82	0.77	0.88

German: The best combination of feature selection and undersampling methods was the univariate feature selection (for 500 features) with random undersampling, as shown in Table 6. The univariate 500 run with random undersampling for 30 minutes (G1) yielded the highest F . This F was 0.07 higher than Wiegand et al.’s F and 0.06 higher than our baseline model’s F . The individual class R scores for the G1 run were: 0.74 for “other,” 0.54 for “insult,” and 0.65 for “abuse.” The univariate 500 run with random undersampling for two hours (G3) yielded the highest R which was 0.16 better than Wiegand et al.’s top model’s R and 0.14 better than our baseline’s R . The individual class R for the G3 run were: 0.69 for “other,” 0.61 for “insult,” and 0.64 for “abuse.” The other two tests included were univariate feature selection for 1,000 features with random undersampling for 30 minutes (G2) and two hours (G4); these runs had close but slightly lower R than the G1 and G3 runs. It is important to note that Wiegand et al. [5] utilized four classes; meanwhile, we used only three, which could have had an impact on their lower F and R in comparison to our results. As stated earlier, we combined Wiegand et al.’s “profanity” class into the “other” class because the data-set held only 1.3% for the profanity class. The baseline model proved to have the highest A and P , because it focused on high A for the majority class while focusing less on the minority classes. For our purposes, as stated before, we focused on a high R for each class and a high R overall. We concluded that the G3 run gave us the best R result when compared to our baseline and Wiegand et al.’s results.

Table 6. Top results from combining feature selection and undersampling methods with H2O Auto-ML for the German data-set for the macro-average \mathcal{P} , \mathcal{R} , \mathcal{F} , and \mathcal{A} . We compared our metrics to the top-performing ML model’s metrics from Wiegand et al. [5] and our baseline run’s metrics with neither feature selection nor undersampling.

<i>German</i>	\mathcal{P}	\mathcal{R}	\mathcal{F}	\mathcal{A}
Wiegand et al. [5]	0.57	0.49	0.53	0.74
Baseline	0.67	0.51	0.54	0.75
Uni. 500 + Random + 0.5h (G1)	0.58	0.64	0.60	0.70
Uni. 1,000 + Random + 0.5h (G2)	0.56	0.63	0.58	0.68
Uni. 500 + Condensed + 2h (G3)	0.57	0.65	0.59	0.67
Uni. 1,000 + Condensed +2h (G4)	0.56	0.63	0.58	0.68


5 Conclusion

Through our paper, we present Auto-ML as a promising tool for abusive language detection in English and German. Once we tackled the imbalanced data problem for our two data-sets by using feature selection and undersampling methods, the *H2O Auto-ML* framework performed as well as or better than the state of the art ML models, Davidson et al. [4] for English and Wiegand et al. [5] for German. Our research is the first of its kind to apply Auto-ML to the problem of abusive language detection online in multiple language data-sets. We conclude that Auto-ML performs as well as human actors selecting an ML algorithm and hyperparameters with which to train while reducing the time and manual complexity of training an ML model.

Given the competitive results of Auto-ML classifiers in comparison to manually trained models, Auto-ML appears to be a promising pathway for further exploration. Especially since the lack of ML expertise and the immense investments required for ML projects are some of the most pressing inhibitors, Auto-ML might be an auspicious approach to comprehensively tackle the curse of abusive language online. This also goes beyond the traditional algorithmic focus of the domain and acknowledges the struggle of smaller companies to get access to “industry-grade” support tools required to tackle the growing burden of malicious online comments.

While our work marks a first step in this direction, further research is required to identify the full potential and necessary changes to Auto-ML to reliably identify abusive language. Important next steps include but are not limited to: Testing further parameter configurations, extending the level of automation to other ML pipeline stages (i.e., data pre-processing, feature selection, or sampling methods) and assessing further Auto-ML frameworks for their suitability.

6 Acknowledgements

The research leading to these results received funding from the federal state of North Rhine-Westphalia and the European Regional Development Fund (EFRE.NRW 2014-2020),  **MODERAT!** Project: (No. CM-2-2-036a).

References

1. Krikorian, R.: New Tweets per second record, and how! (2013), https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how.html
2. ADL: Anti-Semitic Targeting of Journalists during the Presidential Campaign. Tech. rep. (2016)
3. Dhrodia, A.: Unsocial Media: The Real Toll of Online Abuse against Women (2017), <https://medium.com/amensty-insights/unsocial-media-the-real-toll-of-online-abuse-against-women-37134ddab3f4>

4. Davidson, T., Warmusley, D., Macy, M., Weber, I.: Automated Hate Speech Detection and the Problem of Offensive Language. In: *Elev. Int. aaai Conf. web Soc. media (ICWSM 2017)*. pp. 512–515 (2017)
5. Wiegand, M., Siegel, M., Ruppenhofer, J.: Overview of the GermEval 2018 Shared Task on the Identification of Offensive Language. In: *GermEval 2018, 14th Conf. Nat. Lang. Process. (KONVENS 2018)*. pp. 1–10. No. September (2018)
6. Myung, K.M., Yoo, J., Kim, S.W., Lee, J.H., Hong, J.: Autonomic Machine Learning Platform. *Int. J. Inf. Manage.* (2019)
7. Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y., Chang, Y.: Abusive Language Detection in Online User Content. In: *25th Int. Conf. world wide web*. pp. 145–153. IW3C2 (2016)
8. Waseem, Z., Hovy, D.: Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter. *Proc. NAACL Student Res. Work.* pp. 88–93 (2016)
9. Chatzakou, D., Kourtellis, N., Blackburn, J., De Cristofaro, E., Stringhini, G., Vakali, A.: Mean Birds: Detecting Aggression and Bullying on Twitter. *Proc. 2017 ACM web Sci. Conf.* pp. 13–22 (2017)
10. Balaji, A., Allen, A.: *Benchmarking Automatic Machine Learning Frameworks* (2018)
11. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) *Autom. Mach. Learn. Methods, Syst. Challenges*, chap. 4. Springer, Cham (2019)
12. Shepperd, M., Bowes, D., Hall, T.: Researcher Bias: The Use of Machine Learning in Software Defect Prediction. *IEEE Trans. Softw. Eng.* 40(6), 603–616 (2016)
13. H2O.ai: H2O AutoML (August 2019), 3
14. Feurer, M., Springenberg, J.T., Klein, A., Blum, M., Eggenberger, K., Hutter, F.: Efficient and Robust Automated Machine Learning. In: *Adv. Neural Inf. Process. Syst.* pp. 2962–2970 (2015)
15. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) *Autom. Mach. Learn. Methods, Syst. Challenges*, chap. 4. Springer, Cham (2019)
16. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In: *Proc. Genet. Evol. Comput. Conf. 2016*. pp. 485–492. ACM (2016)
17. Friedl, J.: *Mastering Regular Expressions*. "O'Reilly Media, Inc." (2006)
18. Porter, M.F.: An algorithm for suffix stripping. *Program* (2006)
19. Weissweiler, L., Fraser, A.: Developing a stemmer for German based on a comparative analysis of publicly available stemmers. In: *Int. Conf. Ger. Soc. Comput. Linguist. Lang. Technol.* pp. 81–94. Springer, Cham (2017)
20. Bird, S., Klein, E., Loper, E.: *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc." (2009)
21. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, 2825–2830 (2011)
22. Hutto, C., Gilbert, E.: Vader: A parsimonious rule-based model for sentiment analysis of social media text. In: *Eighth Int. AAAI Conf. Weblogs Soc. Media*. pp. 216–225 (2014)
23. Loria, S.: *textblob Documentation*. Tech. rep. (2017)

24. Kumar, V., Sonajharia, M.: Feature Selection: A literature Review. *Smart Comput. Rev.* 4(3), 211–229 (2014)
25. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Gramfort, A., Niculae, V., Prettenhofer, P., Grobler, J., Layton, R., VanderPlas, J., Varoquaux, G., Joly, A., Holt, B.: API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Work. Lang. Data Min. Mach. Learn.* pp. 108–122 (2013)
26. Weiss, G.M.: Mining with Rarity: A Unifying Framework. *ACM SIGKDD Explor. Newsl.* 6(1), 7–19 (2004)
27. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, P.W.: SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res.* 16, 321–357 (2002)
28. Lemaître, G., Nogueira, F., Aridas, C.K.: Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research* 18(17), 1–5 (2017)
29. Forman, G.: An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *J. Mach. Learn. Res.* 3, 1289–1305 (2003)