Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning

Christoph Stanik, Marlo Haering and Walid Maalej
University of Hamburg
Hamburg, Germany
{stanik, haering, maalej}@informatik.uni-hamburg.de

***Abstract*—With the rise of social media like Twitter and of software distribution platforms like app stores, users got various ways to express their opinion about software products. Popular software vendors get user feedback thousandfold per day. Research has shown that such feedback contains valuable information for software development teams such as problem reports or feature and support inquires. Since the manual analysis of user feedback is cumbersome and hard to manage many researchers and tool vendors suggested to use automated analyses based on traditional supervised machine learning approaches. In this work, we compare the results of traditional machine learning and deep learning in classifying user feedback in English and Italian into problem reports, inquiries, and irrelevant. Our results show that using traditional machine learning, we can still achieve comparable results to deep learning, although we collected thousands of labels.**

***Index Terms*—Data-Driven Requirements, Data Mining, Social Media Analytics, Machine Learning, Deep Learning**

I. Introduction

Motivation. Research has shown the importance of extracting requirements related information from user feedback to improve software products and user satisfaction [32]. As user feedback on social media or app stores can come thousandfold daily, a manual analysis of that feedback is cumbersome [31]. However, analyzing this feedback brings opportunities to understand user opinions better because it contains valuable information like problems users encounter or features they miss [31], [12]. Researchers have applied supervised machine learning to filter noisy, irrelevant feedback and to extract requirements related information [27], [14]. Most related works rely on traditional machine learning approaches, which require domain experts to represent the data with hand-crafted features. In contrast, end-to-end deep learning approaches automatically learn high-level feature representations from raw data without domain knowledge, achieving remarkable results in different classification tasks [11], [33], [38].

Objective. In this work, we aim at understanding if and to what extent deep learning can improve state-of-the-art results for classifying user feedback into problem reports, inquiries, and irrelevant. We focus on these three categories because practitioners seek for automated solutions to filter noisy feedback (irrelevant), to identify and fix bugs (problem reports), and to find feature requests as inspiration for future releases (inquiries) [27]. We consider all user feedback as problem reports, that state a concrete problem related to a software product or service (e.g., "Since the last update the app crashes upon start"). We define inquires as user feedback that asks for either new functionality, an improvement, or requests information for support (e.g., "It would be great if I could invite multiple friends at once"). We consider user feedback as irrelevant if it does not belong to problem reports or inquires (e.g., "I love this app").

To fulfill our objective, we employ supervised machine learning fed with crowd-sourced annotations of 10,000 English and 15,000 Italian tweets from telecommunication Twitter support accounts, and 6,000 annotations of English app reviews. We apply best practices for both machine learning approaches (traditional and deep learning) and report on a benchmark.

Preliminary results. Our preliminary results show that, within our setting, traditional machine learning can achieve comparable results to deep learning. One possible explanation is that domain experts' knowledge in traditional machine learning brings considerable performance improvements using simple but powerful features, including specific keywords. In general, the classification of irrelevant user feedback achieves the best results meaning that practitioners could use our reported models to filter noisy feedback.

Contribution. The contribution of this paper is threefold. First, we give insights on how traditional machine learning compares to deep learning on classifying feedback by describing both approaches and by performing a large series of experiments. Second, we provide a replication package containing the scripts and experiment setups. Third, we report the configurations of top-performing machine learning models.

Structure. In Section II, we introduce the methodology of this paper by detailing our research questions, design, and data. Section III describes the pipeline and the setup for both machine learning approaches. Section IV reports on our classification benchmark showing the accuracy and the configuration of the top-performing models. Then, Section V discusses the implications of the results and possible application fields, as well as the threats to validity. Section VI summarizes the related work while Section VII concludes the paper.

II. Methodology

We discuss the research questions, as well as our study design, and the data our analysis rely on.

A. Research Question

The goal of this work is to identify the top-performing model to classify user feedback (tweets and app reviews) into problem reports, inquires, and irrelevant by comparing the traditional machine learning approach with deep learning. We, therefore, state the following research questions:

- **RQ1.** To what extent can we extract problem reports, inquires, and irrelevant information from user feedback using traditional machine learning?
- **RQ2.** To what extent can we extract problem reports, inquires, and irrelevant information from user feedback using deep learning?
- **RQ3.** How do the results of the traditional machine learning approach and the deep learning approach compare and what can we learn from it?

B. Study Design

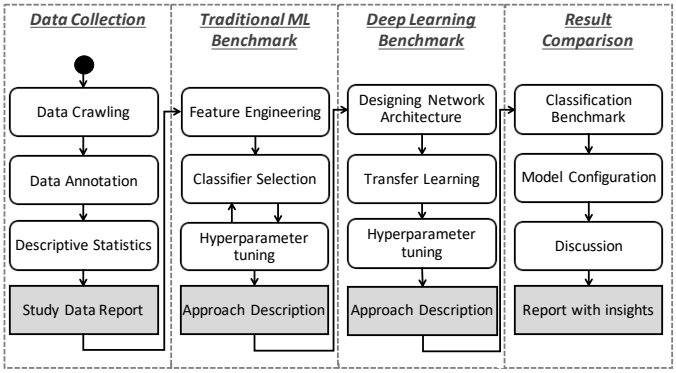

Fig. 1: Overview of the study design.

Figure 1 shows the overall study design. Each white box within the four columns describes a certain step that we performed while the grey boxes show the result that each column produced. The first part of the paper is about the *Study Data*, which we describe later in this section. In the second part, *Traditional Approach*, we perform traditional machine learning engineering, including feature engineering and hyperparameter tuning. In the part *Deep Learning Approach*, we design a convolutional neural network architecture, apply transfer learning for the embedding layer, and finally evaluate the fine-tuned models. In the fourth part *Result Comparison*, we report on the results of our classification experiments (benchmark) comparing the traditional and the deep learning approaches.

C. Study Data

We collected about 5,000,000 English and 1,300,000 Italian Tweets addressing Twitter support accounts of telecommunication companies. From that corpus, we randomly sampled ∼10,000 English tweets and ∼15,000 Italian tweets that were composed by users. As the annotation of so many tweets is very time-consuming, we created coding tasks on the crowd-annotation platform *figure eight*[1]. Before starting the crowd-annotation, we first wrote a coding guide to describe our understanding of problem reports, inquiries, and irrelevant tweets with the help of the innovation center of a big Italian telecommunication company. Second, we run a pilot study to test the quality of the coding guide and the annotations received. Both coding guides were either written or proof-read by at least two native speakers, and we required that the annotators are natives in the language. Each tweet can belong to exactly one of the before mentioned classes and is annotated by at least two persons, three in case of a disagreement. As for the annotated app reviews, we rely on the data and annotations of Maalej et al. [27]. Table I summarizes the annotated data for both languages.

TABLE I: Overview of the study data.

	App Reviews	Tweets	
	English	English	Italian
n_problem_report	1.437	2.933	3.414
n_inquiry	1.100	1.405	2.594
n_irrelevant	3.869	6.026	9.794
TOTAL	6.406	10.364	15.802

Replication package. To encourage replicability, we uploaded all scripts, benchmark results, and provide the annotated dataset upon request[2].

III. Machine Learning Pipelines

We describe how we performed the machine learning approaches and explain certain decisions such as for the selected features. To ensure a fair comparison between the traditional and the deep learning approach, we used not only the same datasets but also the same train and test sets.

A. Traditional Machine Learning

1) Preprocessing: We preprocessed the data in three steps to reduce ambiguity. Step 1 turns the text into lower case; this reduces ambiguity by normalizing, e.g., "Feature", "FEATURE", and "feature" by transforming it into the same representation "feature". Step 2 introduces masks to certain keywords. For example, whenever an account is addressed using the "@" symbol, the account name will be masked as "account". We masked account names, links, and hashtags. Step 3 applies lemmatization, which normalizes the words to their root form. For example, words such as "see", "saw", "seen', and "seeing" become the word "see".

2) Feature Engineering: Feature engineering describes the process of utilizing domain knowledge to find a meaningful data representation for machine learning models. In NLP it encompasses steps such as extracting features from text, as well as selection and optimization. Table II summarizes the groups of features, their representation, as well as the number of features we extracted for that feature group. For instance, the table shows that the feature group "keywords" consists of 37 keywords for the Italian language, each of them being 1 if that keyword exists or 0 if not.

[1]https://www.figure-eight.com/

[2]https://mast.informatik.uni-hamburg.de/replication-packages/

TABLE II: Extracted features before scaling. If not further specified, the number of features applies to all data sets.

Feature Group	Value Boundaries	Number of Features
n_words	$\mathbb{N}$	1
n_stopwords	$\mathbb{N}$	1
sentiment$_{neg}$	$\{x \in \mathbb{Z} \mid -5 \leq x \leq -1\}$	1
sentiment$_{pos}$	$\{x \in \mathbb{N} \mid 1 \leq x \leq 5\}$	1
keywords	$\{0, 1\}$	37 (IT), 60 (EN)
POS tags	$\mathbb{N}$	18 (IT), 16 (EN)
tense	$\mathbb{N}$	4 (IT), 2 (EN)
tf-idf	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	665 (app reviews, EN) 899 (tweets, EN) 938 (tweets IT)
fastText	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	300
TOTAL		1.047 (app reviews, EN) 1.281 (tweets, EN) 1.301 (tweets IT)

We extracted the *length (n words)* of the written user feedback as Pagano and Maalej [31] found that most irrelevant reviews are rather short. One example for such a category is *rating*, which does not contain valuable information for developers as most of the time, such reviews are only praise (e.g., "I love this app."). Excluding or including *stop words*, in particular in the preprocessing phase is highly discussed in research. We found papers that reported excluding stop words as an essential step (e.g., [16]), papers that leveraged the inclusion of certain stop words (e.g., [20]), and others that tested both (e.g., [27]). However, the decision for exclusion and inclusion depends on the use case. We decided to use them as a feature by counting their occurrence in each document.

Further, we extracted the *sentiment* of the user feedback using the sentistrength library [35]. We provide the full user feedback (e.g., a tweet) as the input for the library. The library then returns two integer values, one ranging from -5 to -1 indicating on how negative the feedback is, the other ranging from +1 to +5 indicating how positive the feedback is. The sentiment can be an important feature as users might write problem reports in a neutral to negative tone while inquiries tend to be rather neutral to positive [16], [31], [27]. *Keywords* have proven to be useful features for text classification [36], [27], [18] as their extraction allows input of domain experts' knowledge. However, keywords are prone to overfit for a single domain and therefore might not be generalizable. In this work, we use the same set of keywords for the English app reviews and tweets. We extracted our set of keywords by 1) looking into related work [19], [36], [27], and 2) by manually analyzing 1,000 documents from the training set of all three datasets following the approach from Iacob and Harrison [19]. Kurtanović and Maalej [24], [23] successfully used the counts of *Part-of-speech (POS) tags* for classification approaches in requirements engineering. Therefore we also included them in our experiments.

Maalej et al. [27] successfully utilized the *tenses* of sentences. This feature might be useful for the classification as users write problem reports often in the past or present tense, e.g., "I updated the app yesterday. Since then it crashes." and inquiries (i.e., feature requests) in the present and future tense, e.g., "I hope that you will add more background colors". When extracting the tense using spaCy[3] the Italian language model supported four tenses while for the English language we had to deduce the tense by extracting the part-of-speech tags. *Tf-idf (term frequency-inverse document frequency)* [34] is a frequently used technique to represent text in a vector space. It increases proportionally to the occurrence of a term in a document but is offset by the frequency of the term in the whole corpus. Tf-idf combines term frequencies with the inverse document frequency to calculate the term weight in the document.

FastText [21] is an unsupervised approach to learn high-dimensional vector representations for words from a large training corpus. The vectors of words that occur in a similar context are close in this space. Although the fastText library provides pre-trained models for several languages, we train our own domain-specific models based on 5,000,000 English app reviews, 1,300,000 Italian tweets, and 5,000,000 Italian tweets. We represent each document as the average vector of all word vectors of the document, which is also a 300-dimensional vector. We chose fastText for our word embedding models as it composes a word embedding from subword embeddings. In contrast, word2vec [29] learns embeddings for whole words. Thereby, our model is able to 1) recognize words that were not in the training corpus and 2) capture spelling mistakes, which is a typical phenomenon in user feedback.

3) Experiment Configuration: For the experiment setup, we tried to find the most accurate machine learning model by varying five dimensions (no particular order). In the *first dimension* we target to find the best-performing features of Table II by testing different combinations. In total, we tested 30 different feature combinations such as "sentiment + fastText" and "n_words + keywords + POS tags + tf-idf".

The *second dimension* is testing the performance of (not) applying feature scaling. Tf-idf vectors, for example, are represented by float numbers between 0 and 1, while the number of words can be any number greater than 0. This could lead to two issues: 1) the machine learning algorithm might give a higher weight to features with a high number meaning that the features are not treated equally. 2) the machine learning model could perform worse if features are not scaled.

In the *third dimension*, we perform Grid Search [2] for hyper-parameter tuning. In contrast to Random Search, which samples hyper-parameter combinations for a fixed number of settings [1], Grid Search exhaustively combines hyperparameters of a defined grid. For each hyper-parameter combination in the Grid Search, we perform 5-fold cross-validation of the training set. We optimize the hyperparameters for the f1 metric to treat precision and recall as equally important.

The *fourth dimension* checks whether sampling (balancing) the training data improves the overall performance of the classifiers. For unbalanced data the machine learning algorithm might tend to categorize a document as part of the majority

[3]https://spacy.io/

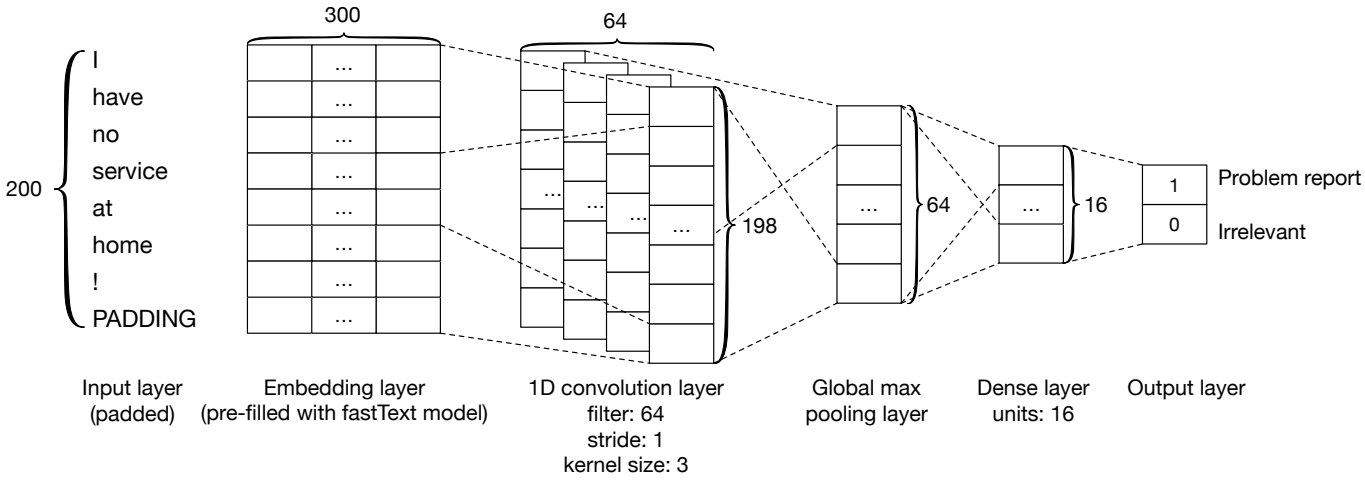

Fig. 2: Neural network architecture for the classification.

class as this is the most likely option. In this work we test both, keeping the original distribution of documents per class and applying random under-sampling on the majority class to create a balanced training set.

Finally, the *fifth dimension* is about testing different machine learning algorithms. Similar to our reasoning for the feature selection, we tested the following algorithms frequently used in related work: Decision Tree, Random Forest, Naive Bayes, and Support Vector Machine [27], [14], [37]. As for the classification, we follow the insights from Maalej et al. [27] and employ binary classification (one for each: problem report, inquiry, and irrelevant) instead of multiclass classification.

B. Deep Learning

1) Deep Learning: Traditional classification approaches require a data representation based on hand-crafted features, which domain experts deem useful criteria for the classification problem at hand. In contrast, neural networks, which are used in deep learning approaches, use the raw text as an input, and learn high-level feature representations automatically [11]. In previous work, researchers applied them in diverse applications with remarkable results to different classification tasks, including object detection in images, machine translation, sentiment analysis, and text classification tasks [6]. However, neural networks are not a silver bullet, and they have also achieved only moderate results in the domain of software engineering [13], [9], [10].

2) Convolutional Neural Networks: Although convolutional neural networks (CNNs) have mainly been used for image classification tasks, researchers also applied them successfully to natural language processing problems [22], [26]. In most cases, deep learning approaches require a large amount of training data to outperform traditional approaches. Figure 2 shows the architecture of the neural network that we used for the experiments in this study. Häring et al. [18] used this model to identify user comments on online news sites that address either the media house, the journalist, or the forum moderator. They achieved promising results that partly outperformed a traditional machine learning approach. The input layer requires a fixed size for the text inputs. We choose the size 200, which we found appropriate for both the app review and the Twitter dataset as tweets are generally shorter and we identified less than 20 app reviews that exceed 200 words. We cut the part, which is longer than 200 words and pad shorter input texts, so they reach the required length. After the input layer our network consists of an embedding layer, a 1D convolution layer, a 1D global max pooling layer, a dense layer, and a concluding output layer with a softmax activation. For the previous layers, we used the tanh activation function. During training, we froze the weights of the embedding layer, whereby 15,000 trainable parameters remain.

3) Transfer Learning: Transfer learning is a method often applied to deep learning using models pre-trained on another task [11]. In natural language processing, a common application of transfer learning is to reuse word embedding models, e.g. word2vec [29] or fastText [21], which were previously trained on a large corpus to pre-initialize the weights of an embedding layer. We applied transfer learning to pre-initialize our embedding layer with three different pre-trained fastText models [21]. During training, we froze the weights of the embedding layer.

4) Hyperparameter Tuning: The network architecture and the hyperparameter configuration can be a crucial factor for the performance of the neural network. Therefore we compared variations of both our CNN architecture as well as training parameters and evaluated the best-performing model on the test set. We performed a grid search and varied the number of filters and the kernel size of the 1D convolutional layer, the number of units for the dense layer, the number of epochs and the batch size for the training, and the number of units for the final dense layer. Due to the small size of our training set, we conducted a stratified 3-fold cross-validation on the training set for each hyperparameter configuration to acquire reliable results. Subsequently, we evaluated the model with the best-

TABLE III: Classification benchmark for the traditional machine learning approach (Trad.) and the deep learning approach (DL). The best f1 score per classification problem and dataset is marked in bold font.

		app review EN				tweet EN				tweet IT			
		p	r	f1	auc	p	r	f1	auc	p	r	f1	auc
Trad.	problem report	.83	.75	**.79**	.85	.46	.82	**.59**	.72	.51	.88	**.65**	.83
	inquiry	.68	.76	.72	.85	.32	.70	**.43**	.73	.47	.82	**.60**	.82
	irrelevant	.88	.89	**.89**	.86	.73	.75	**.74**	.69	.78	.89	**.83**	.73
DL	problem report	.46	.60	.52	.82	.51	.42	.46	.74	.62	.57	.59	.84
	inquiry	.69	.79	**.74**	.94	.40	.40	.40	.75	.51	.57	.54	.83
	irrelevant	.78	.93	.85	.90	.74	.70	.72	.75	.85	.77	.81	.86

performing hyperparameter configuration on the test set. We trained the models with seven epochs and a batch size of 32. We used the Python library Keras [5] for composing, training, and evaluating the models.

IV. Results

In this section, we describe and discuss the results of the classification experiments. We first explain the evaluation metrics. Then, we report on the benchmark in Table III showing the top accuracy. Finally, we explain the configuration of the models leading to the best results from Table IV.

For this work, we report on the classification metrics *precision*, *recall*, and *f1* as presented in related work [16], [36], [27]. For the calculation of these metrics we used sklearn's strictest parameter setting *average=binary*, which is only reporting the result for classifying the true class. Additionally, we report on the Area Under the Curve *AUC* value, which is considered a better metric when dealing with unbalanced data as it is independent of a certain threshold for binary classification problems. In machine learning, Area Under the Receiver Operating Characteristics *ROC AUC* is a metric frequently used to address class imbalance. Davis and Goadrich [7] argue that Precision-Recall AUC (PR AUC) is a more natural evaluation metric for that problem. We optimized and selected the classification models based on f1, the harmonic mean of precision and recall. Thereby either precision or recall can have a rather low value compared to the other.

Table III shows the classification results of the best model for each of the three data sets and each classification problem. For the English app reviews, traditional machine learning generally performs better than deep learning when considering the f1 score. One reason for this difference might be, that for the app reviews, we have only about 6,000 annotated data points while for tweets we have about 10,000 for English tweets and about 15,000 for Italian tweets. For the English tweets, both approaches perform quite similar. While the f1 score seems to be lower for the deep learning approach, the AUC values are similar for both approaches. The results for the Italian tweets show when optimizing towards f1, that deep learning reaches a higher precision, while the traditional approaches achieve a higher recall. The f1 score reveals again that both approaches perform similarly. Based on our results, which are generated by a large series of experiments, we cannot say that for our setup, either of the approaches performs better.

V. Discussion

A. Implications of the Results

In this work, we classified user feedback for two languages from two different feedback channels. We found that when considering the f1 score as a measure, traditional machine learning performs slightly better in most of the examined cases. We expect that our approaches can also be applied to further feedback channels and languages, although some features are language-dependent and need to be updated. For example, our deep learning model requires on top of a training set a pre-trained word embedding model for each language such as the English and Italian fastText models used. Word embeddings capture the similarity between words depending on the domain and language. They are highly adaptable to language development by retraining the model regularly on current app reviews and tweets. It can capture the meaning of transitory terms like Twitter hashtags or emoticons. In traditional approaches, the language-dependent features are keywords, sentiment, POS tags, and the tf-idf vocabulary. This requires more effort for creating models for multiple languages. The rest remains language and domain-independent.

Traditional approaches often perform better on small training sets as domain experts implicitly incorporate significant information through hand-crafted features [4]. We assume that for these experiments, the hand-crafted features derived from the domain experts lead to considerably better classification results. Deep neural networks derive high-level features automatically by utilizing large training samples. We presume that with more training data, a deeper neural network would outperform the traditional approach.

B. Field of Application

Classifying user feedback is an ongoing field in research because of the high amount of feedback companies receive daily. Pagano and Maalej [31] show that, back in 2012, visible app vendors receive, on average, 22 reviews per day in the app stores. Free apps receive a significantly higher amount of reviews (∼37 reviews/day) compared to paid apps (∼7 reviews/day). Popular apps such as Facebook receive about 4,000 reviews each day. When considering Twitter as a data source for user feedback for apps Guzman et al. [12] show that popular app development companies receive on average about 31,000 daily user feedback. Such numbers make it difficult for companies – in particular with popular apps – to employ

TABLE IV: Configuration of the best performing classification experiments for the traditional machine learning and the deep learning approaches. RF = Random Forest, DT = Decision Tree. CNN = Convolutional Neural Network.

Traditional Machine Learning	app review EN	problem report	RF(max_features:None, n_estimators:500). **features**:sentiment, tfidf, **sampling**:true, **scaling**:false
		inquiry	DT(criterion:gini, max_depth:1, min_samples_leaf:1, min_samples_split:4, splitter:random). **features**:tfidf, keywords, **sampling**:false, **scaling**:false
		irrelevant	DT(criterion:gini, max_depth:8, min_samples_leaf:2, min_samples_split:4, splitter:random). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, tfidf, **sampling**:false, **scaling**:false
	tweet EN	problem report	RF(max_features:auto, n_estimators:1000). **features:**sentiment, tfidf, **sampling**:true, **scaling**:true
		inquiry	DT(criterion:gini, max_depth:1, min_samples_leaf:1, min_samples_split:2, splitter:best). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, tfidf, fastText, **sampling**:true, **scaling**:true
		irrelevant	RF(max_features:none, n_estimators:1000). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, fastText, **sampling**:true, **scaling**:false
	tweet IT	problem report	RF(max_features:log2, n_estimators:1000) **features:**sentiment, n_words,n_stopwords, n_tense, n_pos, tfidf, **sampling**:true, **scaling**:true
		inquiry	DT(criterion:entropy, max_depth:8, min_samples_leaf:10, min_samples_split:6, splitter:random) **features:**n_words,n_stopwords, n_tense, n_pos, keywords, **sampling**:true, **scaling**:false
		irrelevant	DT(criterion:entropy, max_depth:8, min_samples_leaf:8, min_samples_split:2, splitter:random) **features:**sentiment, n_words,n_stopwords, n_tense, n_pos, tfidf, keywords, **sampling**:false, **scaling**:true
Deep Learning	app review EN	problem report	CNN(dense_number_units:32, kernel_size:3, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
	tweet EN	problem report	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:16, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
	tweet IT	problem report	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true

a manual analysis on user feedback [15]. Therefore, gaining a deeper understanding of how to 1) filter noise and 2) how to extract requirements relevant information from user feedback is of high importance [27]. Recent advances in technology and scientific work enable new ways to tackle these challenges.

VI. Related Work

In the paper "Toward Data-Driven Requirements Engineering", Maalej et al. [28] describe the concept of *User Feedback Analytics* which contains the two sub-categories *Implicit Feedback* and *Explicit Feedback*. While *Implicit Feedback* deals with usage data such as click events that are collected via software sensors on, e.g., a mobile device, *Explicit Feedback* is concerned with written text such as app reviews. We focus on *Explicit Feedback*, which in the field of requirements engineering often includes either *app reviews* [17], [16], [27], *tweets* [14], [37], product reviews such as *Amazon reviews* [24], [25], a combination of reviews and product descriptions [20], or a combination of platforms [30]. User feedback is essential to practitioners, as it contains valuable insights such as bug reports and feature requests [31]. The classification of user feedback [27] was a first step towards extracting such information. Further studies [24], [25] looked at classified feedback to analyze and understand user rationale—the reasoning and justification of user decisions, opinions, and beliefs. Once a company decides to integrate, for example, an innovative feature request in the software product, it will be forwarded to the release planning phase [36]. In this work, we focus on the classification of user feedback of app reviews and tweets.

App Review Classification. Maalej et al. [27] present experiments on classifying app reviews from the Google Play Store and the Apple AppStore using traditional machine learning. In contrast to their work, we also apply deep learning, included tweets and work with two different languages (English and Italian). Chen et al. [3] introduce *AR-Miner*, a framework focusing on mining and ranking techniques to extract valuable information for developers following the idea of reducing manual effort. Dhinakaran et al. [8] perform app review classification and enhance existing approaches with active learning to reduce the annotation effort for experts.

Tweet Classification. Guzman et al. [14] and Williams and Mahmoud [37] present studies that assess the technical value of tweets for software requirements. Williams and Mahmoud [37] conclude that—after analyzing 4,000 tweets manually—about 51% of the tweets contain technical information useful for requirements engineering. Similarly, Guzman et al. [14] show that about 42% of their 1,350 manually analyzed tweets contain either bug reports, feature shortcomings, or feature requests. Conceptually, both studies follow similar goals and structure by: first preprocessing the data; second classifying tweets into their specified categories; and third grouping similar tweets. Guzman et al. [14] go one step further and present a weighted function to rank tweets by their relevance. Compared to both papers, we have a strong focus on reporting feature engineering by testing diverse features and feature combinations (see Table II). Further, we perform the classification on two different languages and employ deep learning as an addittonal experiment.

VII. Conclusion

In this study, we present a series of classification experiments to find requirements-relevant information in English app reviews as well as in English and Italian tweets. We applied supervised machine learning and compared traditional machine learning and deep learning approaches. We rely our results on a) 10,000 English and 15,000 Italian annotated tweets from telecommunication Twitter support accounts, and b) on 6,000 annotations of English app reviews. Our results show that, within our setting, traditional machine learning can achieve comparable results to deep learning, although we collected thousands of annotations for each channel.

Acknowledgement

The work presented in this paper was conducted within the scope of the Horizon 2020 project OpenReq, which is supported by the European Union under the Grant Nr. 732463. The work was also supported by the "Forum4.0" project as part of the ahoi.digital funding line. We thank Davide Fucci for helping collect and analyze the Italian tweets.

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[2] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[3] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, pages 767–778. ACM, 2014.

[4] F. Chollet. *Deep learning with Python*. Manning Publications, 2018.

[5] F. Chollet et al. Keras. https://keras.io, 2015.

[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[7] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[8] V. T. Dhinakaran, R. Pulle, N. Ajmeri, and P. K. Murukannaiah. App review analysis via active learning: Reducing supervision effort without compromising classification accuracy. *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 170–181, 2018.

[9] S. Fakhoury, V. Arnaoudova, C. Noiseux, F. Khomh, and G. Antoniol. Keep it simple: Is deep learning good for linguistic smell detection? In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 602–611. IEEE, 2018.

[10] W. Fu and T. Menzies. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 49–60. ACM, 2017.

[11] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[12] E. Guzman, R. Alkadhi, and N. Seyff. A needle in a haystack: What do twitter users say about software? In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, pages 96–105. IEEE, 2016.

[13] E. Guzman, M. El-Haliby, and B. Bruegge. Ensemble Methods for App Review Classification: An Approach for Software Evolution (N). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 771–776, Nov. 2015.

[14] E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20. IEEE, 2017.

[15] E. Guzman, M. Ibrahim, and M. Glinz. Prioritizing user feedback from twitter: A survey report. In *2017 IEEE/ACM 4th International Workshop on CrowdSourcing in Software Engineering (CSI-SE)*, pages 21–24. IEEE, 2017.

[16] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 153–162. IEEE, 2014.

[17] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: Msr for app stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 108–111. IEEE Press, 2012.

[18] M. Häring, W. Loosen, and W. Maalej. Who is Addressed in This Comment?: Automatically Classifying Meta-Comments in News Comments. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW):67:1–67:20, Nov. 2018.

[19] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 41–44. IEEE Press, 2013.

[20] T. Johann, C. Stanik, A. M. A. B., and W. Maalej. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 21–30, Sep. 2017.

[21] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[22] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[23] Z. Kurtanović and W. Maalej. Automatically classifying functional and non-functional requirements using supervised machine learning. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 490–495. IEEE, 2017.

[24] Z. Kurtanović and W. Maalej. Mining user rationale from software reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 61–70. IEEE, 2017.

[25] Z. Kurtanović and W. Maalej. On user rationale in software engineering. *Requirements Engineering*, 23(3):357–379, 2018.

[26] M. M. Lopez and J. Kalita. Deep Learning applied to NLP. *arXiv:1703.03091 [cs]*, Mar. 2017. arXiv: 1703.03091.

[27] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik. On the automatic classification of app reviews. *Requirements Engineering*, 21(3):311–331, Sep 2016.

[28] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016.

[29] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] M. Nayebi, H. Cho, H. Farrahi, and G. Ruhe. App store mining is not enough. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 152–154, May 2017.

[31] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 125–134. IEEE, 2013.

[32] F. Palomba, M. Linares-Vasquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pages 291–300. IEEE, 2015.

[33] W. Song and J. Cai. End-to-end deep neural network for automatic speech recognition. *Standford CS224D Reports*, 2015.

[34] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[35] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment strength detection in short informal text. *Journal of the Association for Information Science and Technology*, 61(12):2544–2558, 2010.

[36] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016.

[37] G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10. IEEE, 2017.

[38] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent Trends in Deep Learning Based Natural Language Processing. *arXiv:1708.02709 [cs]*, Aug. 2017. arXiv: 1708.02709.