Proceedings of the ResT-UP2 Proceedings @LREC2022, pages 16–24
Marseille, 20-25 June 2022

A Comparison of Machine Learning Techniques for Turkish Profanity Detection

Levent Soykan, Cihan Karsak, İlknur Durgar Elkahlout, Burak Aytan
TURKCELL
İstanbul, Turkey
{levent.soykan, cihan.karsak, ilknur.durgar, burak.aytan}@turkcell.com.tr

Abstract

Profanity detection became an important task with the increase of social media usage. Most of the users prefer a clean and profanity free environment to communicate with others. In order to provide a such environment for the users, service providers are using various profanity detection tools. In this paper, we researched on Turkish profanity detection in our search engine. We collected and labeled a dataset from search engine queries as one of the two classes: profane and not-profane. We experimented with several classical machine learning and deep learning methods and compared methods in means of speed and accuracy. We performed our best scores with transformer based Electra model with 0.93 F1 Score. We also compared our models with the state-of-the-art Turkish profanity detection tool and observed that we outperform it from all aspects.

Keywords: Profanity Detection, Natural Language Processing, Text Classification

1. Introduction

Profane language generally contains words or phrases that are disrespectful to someone or something. It may include social, sexual, racial insulting contents. The profane language includes vulgar and swear words, obscene expressions, and naughty jokes etc. With the increase of social media use from all age groups, profanity detection is very crucial on social media content and search engines. Most of the service providers and social media platforms are applying detection and masking methods by content moderation to discourage this type of language. The level of profanity detection can be differ from just censoring succ and f words to a sentiment level. Very simply, one can use blacklists consisting of profanity words and search them in the contents. This approach unfortunately does not satisfy the needs as in most of the cases users can find a way to fool these lists buy making on purpose typos, changing the letter by numbers, using different font types or even emojis. Moreover, there are many words that have dual senses than can both express offense and non-offense meaning depending on the context. An alternative way of profanity detection is employing data-driven methods with classical machine learning and deep learning methods. The task is getting harder when someone works with morphologically complex languages like Turkish.

Recently, automatic profanity detection became one of the trending topics in natural language processing. First attempts was focused on hate speech detection (Chen et al., 2012; Davidson et al., 2017; Agarwal and Sureka, 2017). There are several datasets (Kumar et al., 2018; Ibrohim and Budi, 2018) collected for this purpose and even several shared tasks (Zampieri et al., 2020; Zampieri et al., 2019; Basile et al., 2019) are organized for offensive language and hate speech detection.

In this paper, we focus on Turkish profanity detection on search engine queries. Despite the increasing interest on this topic for other languages, there is still very limited research on Turkish. For our best knowledge there is only one available corpus on Turkish offensive language (Çöltekin, 2020) (approximately 40K Twitter entries). (Çelik and Yıldırım, 2020) is conducted a comparison of classical machine learning techniques for Turkish profanity detection. A dataset (approximately 80K) is also collected within this work but the data is not publicly available yet and the only publicly available profanity tool for Turkish for our best knowledge is Sinkaf [1].

The profanity detection task for Turkish search engine queries is challenging in two respects: the first one is the agglutinative structure of Turkish which brings special difficulties such as sparsity problem. The second is the length of the entities because of the nature of search engine queries. The phrases are very short (three words on average) and classifiers should work with a very limited context.

The first aim of this work is to collect large set of data from search engine queries. We collected a corpus of $400K$ entries [2] and labeled them in one of the two profanity classes (True or False mainly). Then we compared the classification performance of several classical machine learning and deep learning techniques on this dataset. This paper is organized as follows: Section 2 introduces the data collection and labeling processes. In Section 3, we explain the data preprocessing steps. Section 4 and 5 focuses classical machine learning and deep learning model setups and their experimental results. Finally we conclude with Section 6.

[1] https://github.com/eonurk/sinkaf

[2] %50 of the data is freely available for academic purposes. Please contact the authors for dataset acquisition.

2. Data Collection

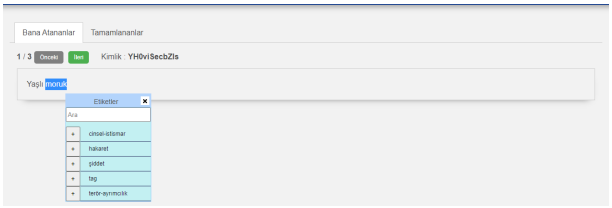
For data-driven profanity detection to perform well on real world scenarios, it is of crucial importance to have training data that has a similar distribution with the real world data. In order to satisfy this constraint, we carried out an extensive data collection and labeling process and collected a dataset of approximately $400K$ phrases/sentences from search engine queries.

2.1. Labeling Process

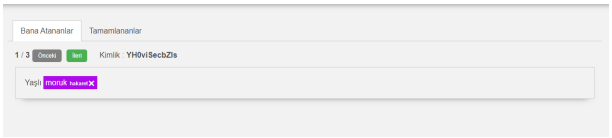
We organized a team of twenty people for data labeling. We shared a labeling guide in which we itemized the tagging criteria and provided positive and negative examples. We grouped offense classes into four as *discrimination*, *sexual abuse*, *profanity* and *violence*. Definition of each class is as follows:

- **Discrimination:** All kind of hate speech that includes unjust or prejudicial treatment of different categories of people, especially on the grounds of race, age, sex, or disability.
- **Sexual Abuse:** All kind of phrases that implies any adulty content including unwanted sexual activity like pedophilia.
- Violence: Phrases involving physical force intended to hurt, damage, or kill someone or something.
- **Succ and f words:** Rude and insulting words or phrases to cause someone to feel hurt, angry, or upset including swear words.

We asked annotators to complete a demo task (100 phrases for each annotator) before initializing the main project. By the help of these demo outputs, we revised the labeling guide and finalized the rule set. During the whole process, we employed an in-house labeling and verification tool. Figure 1 shows an example annotation screen used in the study. In the light of the given instructions, the annotator labeled the word *moruk* (geezer) in the phrase *yaşlı moruk* (old geezer) as profanity word and selected the type as **violence**. At the end of the labeling process, all phrases that are labeled in one of the four classes are recorded as True (Profane class) and the rest is recorded as False(not-Profane).
In the first phase of the annotation process was dedicated to the labeling of the entries and in the second phase was used to check the mismatches between annotators and consolidate the output. Randomly selected 10% percent of the data (equally from each annotator) is considered in the consolidation step in which four experts (different from the annotators) checked the wrong labels and words. If all of the labels are correct, it is selected as the final label. If any of the labels or words is wrong, the true label is determined by the expert. Figure 2 illustrates the consolidation screen used in order to analyze the annotations.



(a) Selecting the Profane Phrase



(b) Final View After Labeling

Figure 1: Labeling Example

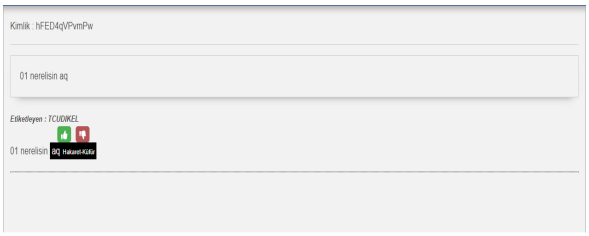


Figure 2: Annotation Verification

2.2. Corpus Statistics

The total dataset we obtained after labeling process consists of **392,806** phrases. In our dataset, each text input is labeled as False (83.6% of the corpus) and True (16.4% of the coprus). Same distribution is maintained during creation of train/test/validation datasets using Stratified Splits & StratifiedKFold. The longest query has 110 words whereas the shortest one is a single word. Looking at the median values, we can state that the most inputs contain three word sequences and average word length is around six characters. Table 1 illustrates the distribution of query lengths. When tokenized by spaces, whole dataset has over $190K$ unique words, including digits, letters, symbols. This means high dimensionality is an important challenge for this study. The data set created in this study will be made partially publicly available with a permissive license.
In this work, we set aside 10% of entire data as test set and performed data analysis and model training on the 90% of the dataset. For the cases that we need a seperate validation set, 10% of train set is used. Validation is mostly used for model performance evaluation and hyperparameter tuning.

3. Data Preprocessing

In order to clean data, we applied several preprocessing step as shown in Figure 3.
We can briefly explain these preprocessing steps as follows:

- **Lowercasing:** We applied lower casing to all texts in the dataset.

	mean	std	min	25%	50%	75%	max
avg. word length	6.02	2.78	0.31	4.8	5.66	7.0	199
word count	3.09	1.66	1.0	2.0	3.0	4.0	110

Table 1: Corpus Statistics

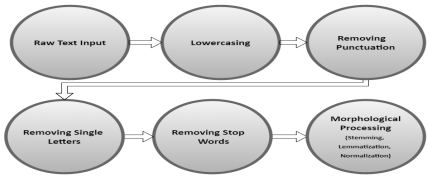


Figure 3: Preprocessing steps applied to the Dataset

processed	normalized	stemmed	lemmatized
181,734	129,532	149,898	151,364

Table 2: Number of word tokens after preprocessing step (Originally: 181,933 tokens)

Figure 4: Word Count of the Dataset

- **Punctuation Removal:** As punctuation has a limited effect on profanity, we removed all punctuations.
- **Single Letter Removal:** Although the dataset includes words from different languages, the dominating language is Turkish. Turkish language does not have single letters as words except *o* (he/she/it) which is already in our stop words list. We removed single letters but as digits & numbers may sometimes indicate offense, we kept numeric values untouched.
- **Stop Words Removal:** We removed the stop words using a pre-defined Turkish Stop Words list. These include written numbers, pronouns (demonstratives such as *bu* (this), *şunlar* (these); possessives such as *onun* (his), *benim* (my, mine), reflexive: *kendim* (myself), *kendin* (yourself)) and helping verbs (*yapmak* (to do), *etmek* (to make), etc.). Note that our decision of removing stop words is based on profanity classification. Removal of stopwords may harm other tasks on Turkish such as summarization & sentiment prediction.
- **Morphological Preprocessing/Spellchecking:** We experimented with three morphological processes **i) Normalization:** Basic spell checker and word suggestion. Noisy text normalization applied with Zemberek(Akin, 2019)), **ii)Stemming:** Only Stemming applied after initial preprocessing Tool with TurkishStemmer (Osman Tunçelli, 2019), **iii)Lemmatization:** Only Lemmatization applied after initial preprocessing Tool with Zeyrek[3]. Each process might change the word completely and may also have adverse results.

3.1. Vector Representation of Text Input

Since machine learning tools accept numeric input, a numeric representation of text is required. Vectorization is a common way of creating numeric features from text data. The most straightforward application is One-Hot-Encoding where all words will be represented as numbers. We used *CountVectorizer* (within scikit-learn) for this step, as it takes care of tokenization and provides n-gram options. Another method is Term Frequency-Inverse Document Frequency (tf-idf), which is based on assigning weights to each token inversely proportional to its frequency across all documents. Tf-idf tokenization helps reduce the effect of stop words/less important words and giving more emphasis to words that are rare and important. By default, vectorization is made using each unique word as a token. However, when grouped together, words might have different meaning or stronger effect. N-Gram Range is a method for grouping all n-word combinations as a single unique token. We represent the comparison of two methods in Section 4.2.

3.2. Feature Selection & Evaluation Metrics

After tokenization and vectorization, our train data has between 120-150K features, and hyperparameter tuning of some ML algorithms with this feature set is inefficient. (especially for tree-based classifiers and ensemble models) For application

[3]https://zeyrek.readthedocs.io/en/latest/

of these models, most relevant features must be identified and data should be represented with fewer features while maintaining the accuracy. To evaluate the dependency of all features and the target variable, we applied Chi2 test, which is a statistical test with the null hypothesis being "the feature and target variable are independent". Test returns a test-statistic and a p-value. For low p-values, we can state that we have enough data to reject the null hypothesis, meaning that the feature is correlated to targets. For our processed data, we have 18.702 unique features that have significant correlation ($p<0.05$) to target. For ensemble models such as RandomForest, this feature set is used.

Since the work might be deployed in profanity detection of online services, a low False Negative rate will be desirable. For this reason, we will use F1 scores mainly as it incorporates both the True-Positive, False-Positive predictions Alongside F1 Score, we will also keep an eye on the recall and precision. Most ML / DL algorithms provide predicted probabilities, where one can also tweak the probability threshold to adjust precision/recall levels. Accuracy could be misleading in imbalanced datasets, but to visualize how a model/pipeline is performing, we can plot a ROC curve. Figure 5 shows a ROC Curve using Logistic Regression. Area under the curve is the main indicator and the diagonal dotted line represents the performance of random guessing where the model cannot distinguish between classes, and as shown in figure, Logistic Regression Model is way above this line.

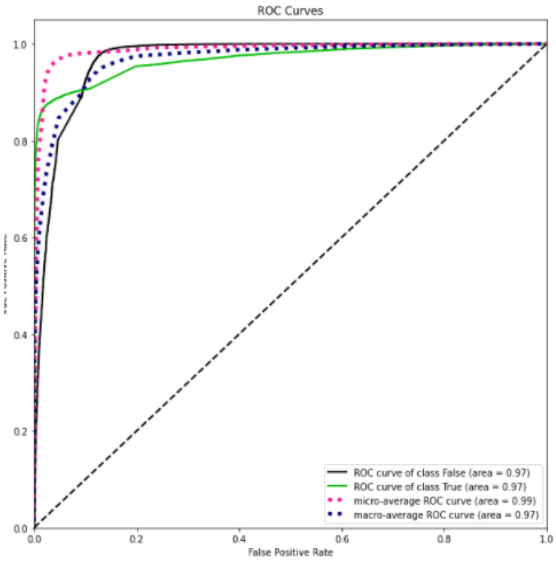


Figure 5: ROC curves for Logistic Regression

4. Classical Machine Learning Experiments

After initial preprocessing, we applied several Machine Learning models to our data.

- **LogisticRegression:** Linear classifier using logistic function (sigmoid curve) to calculate class probabilities. Offers L1-L2 regularization options.
- **SGDClassifier:** A linear classifier applied with Stochastic Gradient Descent where loss is calculated with each sample, and learning rate can be adjusted gradually.
- **LinearSVC:** This model is a faster application of Support Vector Classification with a linear kernel and accepts sparse inputs. The fitting time is much lower compared to standard SVC and is commonly used for text classification.
- **MultinomialNB:** Calculates conditional probability of features, with 'naive' assumption of conditional independence among features.
- **KNeighborsClassifier:** Algorithm based on pre-determined number (k) of nearest data points to each query point.
- **RandomForestClassifier:** Ensemble method fitting a number of Decision Tree Classifiers on sub sample of dataset and averages the outcomes to make predictions. Sub sample size and selection can be controlled with model hyperparameters
- **XGBClassifier:** Uses Gradient Boosting method to combine outputs of a set of Decision Trees where trees are fitted sequentially, and gradient descent is applied for optimization.

4.1. Effect of Morphological Processes

To test the results, we ran the processed data through four ML classification algorithms with default parameters. Table 3 includes the F1-score on 5-fold cross validation of training set with different morphological processes. *Processed* column shows the results with the first four preprocessing steps without morphological preprocessing.

As shown in table 3, it seems that although normalization corrects some spelling errors, it negatively affects total performance, as many misspelled profane words in our dataset are incorrectly changed. The first four preprocessing steps yields good results, lemmatization/stemming can also be tried as they have similar performance. Additional features extracted from text

	processed	normalized	stemmed	lemmatized
SGDClassifier	0.80	0.79	0.82	0.82
LogisticRegression	0.86	0.85	0.87	0.87
LinearSVC	0.91	0.88	0.91	0.91
MultinomialNB	0.81	0.80	0.81	0.81

Table 3: The experimental results with different morphological processes

statistics are not applied, as we did not observe a correlation between these statistics and the target variable. Table 2 shows the final number of unique tokens after preprocessing steps. Figure 4 shows the distribution of word counts.

4.2. Effect of Vectorization

In order to see the effect of vectorization on performance, we performed a cross-validation on LinearSVC model using both Tf-idf & Countvectorizer with Unigram (1,1) and Bi-gram (1,2) options. Table 4 shows the cross validation results. Although the results are close to each other, and CountVectorizer with Unigram seems to work well both in terms of recall and F1-score. Increasing n-gram range does not contribute much, which is somewhat expected as often times profane word can be indicated by a single word. As the performances of ngram-ranges are close, we preferred Counvectorizer with ngram-range(1,1) in the following experiments.

	Cnt(1)	Cnt(2)	Tfidf(1)	Tfidf(2)
Fit T.	7.06	13.25	2.09	3.27
Acc.	0.97	0.97	0.97	0.96
F1	0.91	0.90	0.91	0.89
Recall	0.86	0.84	0.84	0.80

Table 4: Comparison of CountVectorizer and TfidfVectorizer with uni- and bi-grams (fit time in secs)

4.3. ML Experimental Results

Table 5 are the initial results of all models.

As a further step, we applied hyperparameter tuning to LogisticRegression, SGDClassifier, LinearSVC and RandomForestClassifier as shown in Table 6. Best score is achieved by the tuned version of LinearSVC. Classification report and confusion matrix on test set with this model is shown in Table 7. Though there are many False-Positive (Type 1 Error) classifications, the model identifies True (Profane) classes accurately.

5. Deep Learning Methods

After the ML algorithms, we also experimented with deep learning algorithms. We first started with a baseline LSTM model then moved to transformer models BERT and Electra. Finally we tried T5 models.

5.1. Baseline LSTM Model

As a baseline model, we created a two layer LSTM model. We used Keras (Chollet and others, 2015) preprocessing package[4] for preprocessing and Adam (Kingma and Ba, 2015) optimizer, which uses moments calculated with exponentially weighted averages of gradients for optimization. For the loss calculation, we prefered Binary Cross Entropy[5] that uses function uses cross entropy - negative logarithm of predicted probabilities. Our baseline network is created with layers explained in Table 8. The model parameters are as follows:

- Optimizer: AdamW (Learning Rate: 5e-4, weight_decay=1e-3, epsilon: 1e-8)
- GPU Batch Size: 128
- Gradient Clipping: (Max Norm = 2.0)
- Warmup – Linear Schedule with 20.000 steps

5.2. Transformer Models

Transfer Learning is a methodology used in machine learning where the model stores the gained knowledge (updated weights) while training for an objective, and the stored weights are then re-applied with another model to a different problem. This approach is very common in deep learning, where pre-trained models are used and fine-tuned to solve new computer vision and natural language processing problems. The pre-trained models we applied use Transformers(Cho et al., 2014) and Self-Attention(Vaswani et al., 2017) to identify the important parts of text data and learn connections. We use two pre-trained models for our task; BERT(Devlin et al., 2018) & ELECTRA(Clark et al., 2020).

5.2.1. BERT

The main differentiating point of BERT is that it is a Bi-Directional Model. Compared to uni-directional models where context of a word is represented by words to its left (or right), BERT[6] uses context from both sides to represent the words. This is achieved by Masked Language Modelling (MLM)(Song et al., 2019), where

[4] https://www.tensorflow.org/api_docs/python/tf/keras

[5] https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

[6] https://github.com/google-research/bert

model	F1	Precision	Recall	Acc.
LogisticRegression	0.87	0.98	0.78	0.96
SGDClassifier	0.80	0.98	0.68	0.95
KNeighborsClassifier	0.74	0.99	0.60	0.93
LinearSVC	0.92	0.98	0.87	0.98
MultinomialNB	0.84	0.88	0.80	0.95
RandomForestClassifier	0.90	0.98	0.85	0.97
XGBClassifier	0.76	0.99	0.61	0.94

Table 5: ML Algoritms Experimental Results

model	F1	Precision	Recall	Acc.
LogisticRegression	0.89	0.98	0.81	0.97
SGDClassifier	0.82	0.91	0.75	0.95
LinearSVC	0.92	0.98	0.87	0.98
RandomForestClassifier	0.91	0.98	0.85	0.97

Table 6: The Effect of Hyperparameter Tuning

Actual vs. Predict	**True**	**False**
True	5638	125
False	810	32708

Table 7: Confusion Matrix of LinearSVC the Actual Labels vs Predicted Labels

some words are masked in the input and Transformers are used to predict these masked words. As a pre-trained model, Bert has its own vocabulary and word vectors. The vocabulary is fixed, but BERT has a special word piece tokenization[7]. If the word as a whole does not exist in the vocabulary, the Bert tokenizer splits it into several sub-word segments and trains them separately. This method looks really promising for NLP tasks in Turkish, where there are many possible conjugations of each word, due to the agglutinative nature of the language. Application We used *loodos/bert-base-turkish-uncased*[8] model which has 12 encoder layers with over $30K$ tokens and 768 features on every vector.

- Model Class: BertForSequenceClassification
- Optimizer: AdamW (Learning Rate: 1e-5, epsilon: 1e-8)
- GPU Batch Size: 32
- Gradient Clipping: (Max Norm = 1.0)

5.2.2. Electra

Similar to BERT, Electra also uses Transformer mechanisms, but the main difference is about the training part. Instead of MLM, Electra uses Replaced Token Detection as a task for pre-training. In this task, Electra models[9] are trained to distinguish "real" input tokens vs "fake" input tokens generated by another neural network. After pre-training, the generator network is dismissed and model fine-tuning for new tasks are done only with the discriminator. In experimets, We used *dbmdz/electra-base-turkish-cased-discriminator*[10] which is trained on 35GB corpora including Oscar (Abadji et al., 2022) and Opus corporas(Aulamo et al., 2020). We used the model with the following parameters:

- Model Class: ElectraForSequenceClassification
- Optimizer: AdamW (Learning Rate: 1e-5, weight_decay=1e-2, epsilon: 1e-8)
- GPU Batch Size: 32
- Gradient Clipping: (Max Norm = 1.0)
- Loss Function: BinaryCrossEntropy
- Warmup: Linear Schedule with 20.000 steps.

5.3. T5

T5 (Raffel et al., 2019) model is recently used in various NLP tasks including summarization, classification, question answering, etc. Since it is a sequence-to-sequence model, it is available for our task too. During pre-training objective, the model is trained to predict spans of multiple words as well as single word tokens. This helps the model learn sequential relationships and language structure better. The model has

[7] https://huggingface.co/docs/transformers/tokenizer_summary#wordpiece
[8] https://huggingface.co/loodos/bert-base-turkish-uncased
[9] https://github.com/google-research/electra
[10] https://huggingface.co/dbmdz/electra-base-turkish-cased-discriminator

Layer name	Number of Parameters
Embedding(170913, 500)	85M
LSTM(500,64,numlayers=2,batchfirst=True,drpout=0.5)	16K
Linear(infeatures=64,outfeatures=32,bias=true)	256
Linear(infeatures=32,outfeatures=1,bias=true)	256
Sigmoid()	32768
Dropcout(p=0.5, inplace=False)	512

Table 8: Model Summary

a generate method where it generates IDs, which are then transformed to words by the tokenizer. Therefore, in our study, we received the output as text 'True' – 'False' strings and converted them to numeric 0 and 1 afterwards. We used *mt5-small-turkish-question-paraphrasing* [11] model whic is pre-trained on TQP dataset V0.1 (M. Yusuf Sarıgöz, 2021). We used this model with the following parameters:

- Model Class: T5ForConditionalGeneration
- Optimizer: AdaFactor (Learning Rate: 1e-3)
- GPU Batch Size: 32
- Gradient Clipping: (Max Norm = 1.0)
- Warmup: Linear Schedule with 20.000 steps.

5.4. Experimental Results

Table 9 shows the results of fine-tuned models of deep learning methods on our test data. As seen in the table, our baseline method Classifier Network performs similar to LinearSVC but BERT and ELECTRA performs better than all classical machine learning methods explained in Section 4. We also compared the algorithms from the response time aspect, as depending on the use case, one can sacrifice performance for a faster algorithm. Table 10 shows the inference speed[12] of algorithms for 100 samples from our test data. The average text length of this sample is 18 chars.

We also compared our results with the publicly available Sinkaf tool which is implemented with both classical machine learning and deep algorithms. We selected the same algorithms of Sinkaf that we performed best for both categories (LinearSVC and BERT). As seen in the last two columns of the Table 9, our tool outperforms Sinkaf for both cases.

6. Conclusion

In this work, we focused on Turkish profanity detection of search engine entries. In order to build a model that effectively classifies given text sequences as profane or not-profane, we first collected approximately $400K$ data following a labeling process. Later we applied several classical machine learning algorithms and deep learning models. We compared each approach's performance from both accuracy and speed aspects. Although we have slightly better results with BERT & Electra models (F1 score: 0.93), a default LinearSVC model (F1 score: 0.92) also performs closely to transformer models. This strengthens our first indication after n-gram model comparison that identifying a text as profane/not profane is mostly indicated with single words rather than word groups or contextual meaning/clues. Therefore, simple non-sequential, linear algorithms are almost as effective as deep learning networks for classification of profanity detection. Looking at the predicted validation data, Linear Model missed the True labels if the profane word has an uncommon suffix or joined with another word (by mistake or intentionally). Additional recall performance of Pre-trained Transformer models come from these samples, where sub-word tokenization and embedded vectors helped the model classify these texts more correctly.

[11] https://huggingface.co/dbmdz/electra-base-turkish-cased-discriminator

[12] The configuration of the machine: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s) Installed Physical Memory (RAM), 16,0 GB

7. Bibliographical References

Abadji, J., Ortiz Suarez, P., Romary, L., and Sagot, B. (2022). Towards a Cleaner Document-Oriented Multilingual Crawled Corpus. *arXiv e-prints*, page arXiv:2201.06642, January.

Agarwal, S. and Sureka, A. (2017). Characterizing linguistic attributes for automatic classification of intent based racist/radicalized posts on tumblr microblogging website. *CoRR*, abs/1701.04931.

Akin, A. (2019). Zemberek.

Aulamo, M., Sulubacak, U., Virpioja, S., and Tiedemann, J. (2020). OpusTools and parallel corpus diagnostics. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 3782–3789, Marseille, France, May. European Language Resources Association.

Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Rangel Pardo, F. M., Rosso, P., and Sanguinetti, M. (2019). SemEval-2019 task 5: Multilingual detection of hate speech against immigrants and women in Twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 54–63,

model	F1	Precision	Recall	Accuracy
Baseline LSTM	0.92	0.98	0.86	0.97
BERT	0.93	0.96	0.90	0.98
Electra	0.93	0.96	0.89	0.98
T5	0.90	0.94	0.87	0.97
Sinkaf-LinearSVC	0.30	0.75	0.18	0.85
Sinkaf-BERT	0.48	0.80	0.41	0.83

Table 9: Deep Learning Experimental Results

model	mean	std
LinearSVC	171.2ms	8.4ms
Baseline LSTM	231.9ms	17.2ms
BERT	5.0sec	121.8ms
Electra	5.1sec	117.1ms
T5	2.9sec	124.2ms

Table 10: Deep Learning Experimental Results

Minneapolis, Minnesota, USA, June. Association for Computational Linguistics.

Çöltekin, c. (2020). A corpus of turkish offensive language on social media. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 6174–6184, Marseille, France.

Chen, Y., Zhou, Y., Zhu, S., and Xu, H. (2012). Detecting offensive language in social media to protect adolescent online safety. In *International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing, IEEE*, pages 71–80, 09.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Chollet, F. et al. (2015). Keras.

Clark, K., Luong, M., Le, Q. V., and Manning, C. D. (2020). ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555.

Davidson, T., Warmsley, D., Macy, M., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *Proceedings of ICWSM*, 03.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Çelik, A. and Yıldırım, B. (2020). Turkish profanity detection enhanced by artificial intelligence. In *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4.

Ibrohim, M. O. and Budi, I. (2018). A dataset and preliminaries study for abusive language detection in indonesian social media. *Procedia Computer Science*, 135:222–229. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Yoshua Bengio et al., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kumar, R., Reganti, A. N., Bhatia, A., and Maheshwari, T. (2018). Aggression-annotated corpus of Hindi-English code-mixed data. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Osman Tunçelli, Burak Özdemir, H. O. (2019). Turkishstemmer.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T. (2019). MASS: masked sequence to sequence pre-training for language generation. *CoRR*, abs/1905.02450.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., and Kumar, R. (2019). Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 75–86.

Zampieri, M., Nakov, P., Rosenthal, S., Atanasova, P., Karadzhov, G., Mubarak, H., Derczynski, L., Pitenis, Z., and Çöltekin, a. (2020). Semeval-2020 task 12: Multilingual offensive language identification in social media (offenseval 2020). In *Proceedings of the 14th International Workshop on Semantic Evaluation*.

8. Language Resource References

M. Yusuf Sarıgöz. (2021). *monatis/tqp: V0.1 (v0.1) Turkish Question Paraphrasing dataset*. Zenodo, ISLRN https://doi.org/10.5281/zenodo.47198011.