Reproducing Web2Text: Deep Structured Boilerplate Removal, Group 47

Bernhard Steindl
01529136
Austria

Maximilian Zirps
01429041
Austria

Fabian Windbacher
11809633
Austria

ABSTRACT

Web2Text is a deep learning model for removing boilerplate from webpages. Its performance generally exceeded other baseline models. In this work, we reproduce their work. Difficulties in setting up and running their codebase are described and overcome. Also, looking through their code, we find inconsistencies with the paper. Furthermore, we find differences in the results and examine them via statistical significance testing. Importantly, however, we find that the major statements in terms of performance do hold true. Our adjustments and extensions of the original Web2Text repository can be found on GitHub [1].

1 INTRODUCTION

Vogels et al. (2018) propose in *"Web2Text: Deep Structured Boilerplate Removal"* a new model for boilerplate removal resp. content extraction in the context of a single webpage without considering other pages from the same site. NLP and information retrieval applications utilize web page content for building search machine indexes for example and studies show using only the main content of a page increases performance. But, web pages consist not only of main content but also boilerplate / template code, like for example ads, navigation, banners, etc. Vogel et al. introduce with Web2Text a model for boilerplate removal that can classify sections of a webpage as main content or boilerplate.

The model pipeline of Vogel et al. is comprised of 5 steps. First, a raw html webpage is pre-processed by parsing as a Document Object Moel tree (DOM tree), and transformed afterwards to get rid of nodes that are meaningless, like for instance nodes containing only whitespace or no content. The resulting tree-structure is called a Collapsed DOM (CDOM).

Block segmentation is the next step in the pipeline. The authors consider the leaves of the Collapsed DOM tree (leaves strategy) as a sequence of blocks to be labeled as main content or boilerplate.

In the feature extraction step, they extract block and edge features of text blocks from the CDOM. They collect 128 block features that are supposed to represent the content of a block (e.g. "the node is a <p> element" or "average word length"). Edge features describe characteristics between pairs of adjacent text blocks and in total 25 are captured (e.g. tree distance as the sum of the number of hops from both nodes to their common ancestor).

On these features two convolutional neural networks (CNNs) are applied to learn and yield non-linear unary potentials for each text block and pairwise potentials for each pair of neighboring text blocks. The set of potentials define a hidden Markov model. The unary potential represents the probability of a text block being labeled as content or boilerplate. The pairwise potentials are transition probabilities of the labels of a pair of neighboring text blocks and also sum to one. The two CNNs use a different architecture. One CNN receives a sequence of text block features and yields unary potentials for each block. The second CNN receives a sequence of edge features corresponding to the sequence of edges to be labeled and returns pairwise potentials for each block-pair. The unary CNN outputs sequences of 2 values per block and the pairwise network sequences of 4 values per block-pair. For both networks cross-entropy on the respective potentials is used as a loss function and the CNNs are trained separately.

Given a web page, the model infers an optimal labeling sequence for a sequence of text blocks using the CNN potentials set and by utilizing the Viterbi algorithm.

Vogels et al. test Web2Text's boilerplate removal performance on CleanEval 2007, a corpus of manually annotated web pages and a benchmark dataset for this task. They evaluate the model on the original datasets train, dev, test split (55, 5 and 676 pages) and also on a second, custom Web2Text split (531, 58 and 148 pages). They argue to use a larger number of web pages for training because Web2Text has 30,830 parameters (17,960 for unary and 12,870 for pairwise CNN) to train.

The authors of the paper also evaluate their model in the setting of information retrieval using the ClueWeb12 collection of web pages.

A further contribution of the authors is *Automatic Block Labeling*. Existing datasets for the content extraction task consists of pairs of web pages and corresponding manually extracted, cleaned text. But, the alignment between the source text and the cleaned text is missing. Also these datasets lack block-label pairs. Vogel et al. propose an automatic method to recover the block labeling and the alignment between source and cleaned text, given pairs of the original web page and the cleaned text.

In the following sections we will try to reproduce (part of) their results:

(1) We will check their code, whether the implementation matches with their statements in the paper.
(2) We will run their training and inference code.
(3) We will compare results of evaluation and perform statistical singificance tests.

2 REPRODUCTION STEPS

The major task is to reproduce the results that were obtained in the boilerplate removal experiment. We identify the following steps:

(1) Setup workspace

[1] GitHub repository of adjustments and extensions of Web2Text, https://github.com/Bernhard-Steindl/web2text, last accessed on January 30th, 2022

Experiment Design 'WS21, Jan, 2022, Wien, AT

https://doi.org/10.5281/zenodo.5920481

(2) Evaluate the model with pre-trained checkpoints
(3) Train the models
(4) Reproduce experimental results with Web2Text and CleanEval split
(5) Verification of model implementation
(6) Significance testing

The authors have provided and have linked to a GitHub-Repository[2] in their paper, containing not only their source code, but also many intermediate artifacts. Therefore, we can try to reproduce the whole process or parts thereof. We do not reproduce the information retrieval experiments, since they require data in the order of TB. This is not reasonable within the bounds of this project and exceeding the limits of our personal hardware. Furthermore, we limit ourselves to the machine learning part of the process. We do not concern ourselves with the feature extraction in detail.

3 REPRODUCTION

3.1 Setup workspace

Before starting an experiment one has to setup a development workspace first. We could not find any information about which Python version or Python libraries were used in the paper. The GitHub repository does not have a `requirements.txt` file, which usually lists `pip` packages. But the repository has a `README.md` file, which tells us in the Installation section that two steps are required. According to the first step one should install Scala and SBT. The second step in the installation section tells us to install Python, Tensorflow and Numpy.

In the next `README.md` section, Usage, they refer to a blog post by Xavier Geerinck on how to run the code with step-by-step instructions [3]. But at the time of reproducing the experiment the website was not available anymore (HTTP 404 error). However, we found the blog to be still alive but under a different URL [4].

3.1.1 Install Scala-SBT. The authors explicitly state that there code was tested with SBT 1.3.3 and link to the Scala-SBT download webpage [5]. For those not familiar with Scala-SBT, sbt is an interactive build tool for Scala, and Java code, where one defines tasks in Scala, and you can run them in parallel from an interactive shell. The authors also refer to a docker image, named `hseeberger/scala-sbt:8u222_1.3.3_2.13.1`, and provide the following command for running a docker container from that image:

```
docker run -it --rm \
    --mount type=bind,source="$(pwd)",target=/root \
    hseeberger/scala-sbt:8u222_1.3.3_2.13.1 \
    sbt "runMain
    ch.ethz.dalab.web2text.ExtractPageFeatures
   result/input.html result/step_1_extracted_features"
```

In the `README.md` file in section Usage further down, they explain that one should run `ExtractPageFeatures` with sbt, where the argument are the input html file and the desired output base filename (`filename_base`). Apparently, the script outputs two files, `{filename_base}_edge_feature.csv` and `{filename_base}_block_features.csv`. We found the docker container and the corresponding Dockerfile on DockerHub [6] and GitHub [7].

According to the blog post of Geerinck, he also used SBT 1.3.3 on Windows, because 1.3.4 and 1.3.5 have not worked for him. But, he also wrote that he had to change a bat file of sbt on Windows (`C:\ProgramFiles(x86)\sbt\bin\sbt.bat`).

Interestingly, Geerinck has listed on his blog as prerequisite an installed Java version which is greater than 1.8, though the paper authors have mentioned Java neither on their GitHub repository nor in the paper. However, when thinking what the version numbers mean in the docker image name or when looking into the Image layers of the Docker image [8], we find more information regarding version numbers and packages used. These docker image layers show which commands are executed on running the docker container. We found that the authors have used in their docker container a openjdk Java version 1.8 (`8u222b10`) for Linux, probably for the x64 architecture [9]. Which is a contradiction to Geerinck's blog, where he wrote that a Java version greater than 1.8 is required. We found that the SBT version used in the docker container is 1.3.3 and matches the one written on the GitHub repository and Geerinck's blog. We found also that the authors have never mentioned which Scala version they have used, but they just have written in the `README.md` that one should install Scala. Though, we can read the Scala version used in the name of the Docker image or in the Docker image layers. The authors used as Scala version 2.13.1 in the Docker container. But, if we look into the `build.sbt` file of the web2text source code, we can read a different Scala version, namely `scalaVersion := "2.10.4"`, is used in the project.

However, the authors have written down one possibility to use the Docker container, others may not use the docker container, but instead install SBT, Java and Scala globally on their system. Also, the blog page of Geerinck does not mention Scala at all. Then, users may find it hard to figure out which version to install. We guess, not all have the idea to look into a Docker image's layers to find the versions used or can derive by its naming format that 8u222 may be a Java version.

Pulling the docker container for Scala-SBT with `docker pull hseeberger/scala-sbt:8u222_1.3.3_2.13.1` was successful. However, naively running the command provided in the `README.md` file has not yielded success.

```
docker run -it --rm \
  --mount type=bind,source="$(pwd)",target=/root \
  hseeberger/scala-sbt:8u222_1.3.3_2.13.1 \
```

[2] Original web2text GitHub repository — https://github.com/dalab/web2text, last accessed on January 7th, 2022
[3] linked blog post on how to run the code from the GitHub repository — https://xaviergeerinck.com/post/ai/web2text, website was not accessible resp. returned a HTTP 404 error as on January 7th, 2022
[4] Web2Text - Deep Structured Boilerplate Removal - Running the Code by Xavier Geerinck — https://xaviergeerinck.com/post/2020/1/2/ai-ml-web2text, last accessed on January 7th, 2022
[5] http://www.scala-sbt.org/download.html, last accessed on January 7th, 2022
[6] https://hub.docker.com/r/hseeberger/scala-sbt/
[7] https://github.com/hseeberger/scala-sbt
[8] Docker image `hseeberger/scala-sbt/8u222_1.3.3_2.13.1` image layers — https://hub.docker.com/layers/hseeberger/scala-sbt/8u222_1.3.3_2.13.1/images/sha256-fdd752b9515f3f5df984894582360621b0396eef589c82e8e0de954dd4f97414?context=explore, last accessed on January 7th, 2022
[9] Java version (probably) download used in Docker container `hseeberger/scala-sbt/8u222_1.3.3_2.13.1` – https://github.com/AdoptOpenJDK/openjdk8-upstream-binaries/releases/download/jdk8u222-b10/OpenJDK8U-jdk_x64_linux_8u222b10.tar.gz

```
sbt \
"runMain ch.ethz.dalab.web2text.ExtractPageFeatures \
 result/input.html result/step_1_extracted_features"
```

One would get a `java.io.FileNotFoundException` when running this command, because no such file `result/input.html` exists. This docker command runs the scala-sbt docker container and mounts the current working directory on the host machine into the container under `/root`, that is the GitHub repository folder would be available in the container. After that, `sbt` runs `ExtractPageFeatures` and supplies two arguments. We found that one could also just run the docker container without passing the `sbt` command, and within the docker container you can then run the page feature extraction in a bash in a second step. We were not successful in running the feature extraction, training and evaluation for the whole sets of input files within the docker container, because we ran into a `Java.lang.OutOfMemoryError: Java heap space` error and we could not fix it by giving sbt more memory via command line options. Therefore, we continued to use a globally installed sbt version, instead of the docker container. We suggest to only use the docker container when trying feature extraction and classification on a single html input file.

We also tried to install sbt 1.3.3 globally on an Ubuntu 18.04 host and run the `sbt` command without a docker container, as Geerinck suggested on his blog post, which turned out to also yield a result for step 1, the page feature extraction.

3.1.2 Install Python and libraries. The authors have not provided a pip requirements file on their GitHub repository, but tell us in the `REAMDE.md` file to "Install Python 3.7 with Tensorflow 1.15 and NumPy". They have not specified the NumPy version used, though.

According to the blog post of Geerinck, he used the same Tensorflow version 1.15, but he used a NumPy version of 1.18. Geerinck has not written which Python version he used.

We struggled with the undetailed setup process for the Python libraries at first, because we have not found the blog post of Geerinck first, so we had to try manually different versions. Only much later, we found the Geerinck blog post under a different URL. One has to be careful to really use Python 3.7 and Tensorflow 1.15, as newer versions seem to fail. At the time of this report TensorFlow for Python is already out in version 2.7. Geerinck also notes that the original Web2Text code is not up to date with the latest version of Tensorflow and breaks when using newer versions.

For reproduction we created a new Conda environment with Python 3.7 and installed the Python dependencies Geerinck wrote on this blog on our local machines.

```
conda create -n web2text -y python=3.7
pip install numpy==1.18.0 tensorflow==1.15.0 \
    tensorflow-gpu==1.15.0
```

According to the TensorFlow webpage [10] CPU and GPU packages are separate for releases 1.15 and older. When running the `pip install` command on a Mac you get an error when trying to install `tensorflow-gpu`, because there is no TensorFlow GPU build for macOS, so those of our group who have Mac could not install `tensorflow-gpu`.

We also noted that the authors import from the python package `future` in `src/main/python/main.py`, which has to be installed additionally. There was no information about this package or which version to use. We decided to install future==0.18.2.

3.1.3 Install NVIDIA CUDA Toolkit. Training a neural network is usually slow in our experience when only done on a CPU. To get a speed up one uses the support of a NVIDIA GPU with NVIDIA CUDA. Interestingly, the paper authors have not written anything about which NVIDIA CUDA toolkit or NVIDIA cuDNN version they have used. In the blog post of Geerinck, he suggests to install CUDA Toolkit version 10.0 [11] and notes that a version greater 10.0 does not work. Geerinck further suggests to install NVIDIA cuDNN [12] for version 10.0. It is not really clear for us which NVIDIA cuDNN version we should install from that.
Nevertheless, according to the TensorFlow tested build configurations website [13] a TensorFlow version with GPU optimization for version 1.15 is tested with Python 3.7, cuDNN version 7.4 and CUDA 10.0. Therefore, our guess was to install NVIDIA cuDNN version 7.4. However, it turned out that the paper authors have not used cuDNN version 7.4 later on, when trying to run step 2 from the `README.md`, that is the `main.py` file with the `classify` option, which uses an existing model checkpoint. If you would run the script with the wrong cuDNN version, you would get an error from `cuda_dnn.cc:319` saying `"Loaded runtime CuDNN library: 7.4.2 but source was compiled with: 7.6.0"`. After trial and error, we upgraded the NVIDIA cuDNN version from 7.4.2 to 7.6.5, which may have been the version the authors used (probably they used a different patch version).

3.1.4 Setup a notebook in Google Colab. We wrote a notebook for Google Colab. We provide the Google Colab notebook named `word2text_colab.ipynb` in our GitHub repository.

At the time of running the experiment a Google Colab system runs Ubuntu 18.04.5 LTS and offers free access to a NVIDIA GPU (e.g. Tesla K80) if available. In the notebook, we propose to uninstall and remove previous installed NVIDIA, CUDA and cuDNN packages and instead install CUDA 10.0.130-410.48 and libcudnn version 7.6.5.32-1+cuda10.0. We use Python 3.7.12 and install fixed versions for pip libraries, namely numpy==1.18.0, tensorflow==1.15.0, tensorflow-gpu==1.15.0 and future==0.18.2. We note that Google Colab comes with already pre-installed versions of pip libraries, but instead of using them we had to install specific versions the authors used for reproducing the experiment. We install sbt=1.3.3 globally via apt-get, because running a docker container on Colab is not possible to our knowledge. We were not able to install exactly the same Java 8 version as the authors used in their docker container (openjdk 8u222b10), but we managed to install a more up-to-date resp. patched Java 8 version, namely openjdk build 1.8.0_312-8u312-b07-0ubuntu1 18.04-b07. In the notebook we show how to train, evaluate the model and classify a single html file.

[10] TensorFlow webpage Install GPU — https://www.tensorflow.org/install/gpu, last accessed on January 7th, 2022

[11] NVIDIA CUDA Toolkit archive — https://developer.nvidia.com/cuda-toolkit-archive, last accessed on January 7th, 2022

[12] NVIDIA cuDNN download — https://developer.nvidia.com/rdp/cudnn-download, last accessed on January 7th, 2022

[13] TensorFlow tested build configurations — https://www.tensorflow.org/install/source#gpu, last accessed on January 7th, 2022

3.2 Verification of model implementation

After obtaining the block and edge features and converting the result into the `cleaneval.npy` file, one can train or evaluate the model. In this section we are documenting our findings of verifying the source code of the model with the descriptions from the paper. In chapters *3.4 CNN Unary and Pairwise Potentials*, *3.5 Inference* and *4.2 Training Details* of the paper by Vogels et al. they describe the model. The model consists of training two separate CNN models. One has to pass `train_unary` and `train_edge` to `main.py` to train the unary CNN and the pairwise CNN, respectively. Model evaluation is started by passing `test_structured` to `main.py`. In the `main.py` the model is implemented with TensorFlow, but auxiliary scripts used are `forward.py` and `viterbi.py`.

3.2.1 Training of CNNs. We found that the unary CNN features training data shape is [128, 9, 1, 128], which is partly comparable with the papers claim that "each iteration processes a mini-batch of 128 9-text-block long Web page excerpts". The data shape for the pairwise CNN is [128, 8, 1, 25], which is covered by the paper by describing that "edge features capture information on each pair of neighboring text blocks. We collect 25 features for each such pair". Because the pairwise CNN is trained on pairwise text nodes, it makes sense to chose an even number of text blocks, but we found no justification why they have chosen 9 text blocks for the unary net.

We can confirm that the authors have employed dropout regularization with a rate of 0.2. However, we cannot confirm that the authors have used a L_2 weight decay with a rate of 10^{-4} as claimed in the paper. Instead, we found that no L_2 regularization was used. By default the weight decay is set to zero, so no L_2 regularization is added in the code. Vogels et al. have used for convolution weight initialization a truncated normal distribution with a mean of 0 and a standard deviation of 0.1. The authors combined 2-D convolutional layers with a bias, but have not mentioned anything about that in the paper. They initialized the bias weights with zeros in the code. The weight initialization was not documented in the paper. We can confirm that the unary model uses 5 convolutional layers, with filter sizes (50,50,50,10,2), with kernel sizes (1,1,3,3,3) and with ReLU activation functions between layers. Also the papers claim for the pairwise model is true, because it actually uses filter sizes of (50,50,50,10,4), kernel sizes of (1,1,3,3,3) for the Conv layers and ReLu non-linear activations between them. We further found that all conv filters use a stride of the sliding window of 1, as described in the paper. The authors use "SAME" padding, so we can also accept the claim in the paper that they have used zero padding for the convolutional layers. The output of the neural network forward-pass (logits) is resphaped to two values per block for the unary net (batch_size,2) and four values per block pair for the pairwise net (batch_size,4), respectively.

After obtaining the logits, the loss is computed for each network separately, which involves the logits and the response potentials labels. According to the paper the network output is normalized using softmax and the cross-entropy is minimized. We could not verify if the formula presented in the paper actually computes the cross-entropy loss. But, we found that the authors used the Tensorflow function `sparse_softmax_cross_entropy_with_logits`, which computes the softmax followed by computing the cross entropy. The authors have then reduced the cross-entropy result to a single scalar by calculating the mean. We could not find a note in the paper that they have used mean reduction. The code for calculating the loss suggests that the authors have at least experimented with a L_2 regularization, because a regularization penalty is been added to the loss if a regularizer has been added to the model, but as already stated above the current repository version does not use any L_2 regularization. As described by the authors the code uses a learning rate of 10^{-3} and the Adam optimizer for loss minimization.

We can confirm that the authors train for 5,000 steps, but they also perform early stopping. Every 100 steps model validation is carried out. The accuracy, precision, recall and F1 score are computed for the unary model for the predictions on the validation set. For the pairwise model the accuracy is computed. Interestingly, the authors determine whether to save the current model to file based on the F1 validation score of the unary network and the validation accuracy of the pairwise network, respectively, instead of using the validation loss. The authors have merely written in the paper that they perform early stopping by picking the model with the lowest validation set error. But they have not explained the reason for not using the validation loss, but instead using different metrics for the neural networks as an early stopping criteria. We note that the model with the lowest validation loss is not necessarily the model with the lowest F1 score or accuracy, respectively.

3.2.2 Model Evaluation. The evaluation is done on the test set and computes the accuracy, precision, recall and F1 score for the unary model alone and also for the joint structured model consisting of the unary and the pairwise network.

The prediction for the pairwise model is done by forwarding the block and edge features through the two neural networks. The prediction of the class labels of the joint structured model involves the use of the Viterbi algorithm. The Viterbi algorithm is invoked with the unary logits, the pairwise logits and the interpolation factor parameter λ. According to the paper the depicted hidden Markov model is maximized using the Viterbi algorithm to find the optimal labeling given the predicted CNN potentials. The authors also state that hey have used as interpolation factor $\lambda = 0.1$, however the code uses $\lambda = 1$. The file `viterbi.py` implements the Viterbi algorithm. We have not verified the implementation of the algorithm as part of our reproduction task, but we noticed that the function calculates the softmax of the unary and pair-wise potentials in the beginning. We found that the softmax function defined in the file calculates the exponent of the potentials and subtracts the maximum value, before dividing by the sum of values. We cannot explain why the maximum value is subtracted in the softmax calculation, because to our knowledge one just computes the exponent of values and divides by the sum of values. The Viterbi algorithm was not described as part of the paper so we have not verified the code any further. Therefore, we were also not able to verify the expression describing the hidden Markov model.

The prediction for the unary model forwards only the block features through the unary network. For the class label predictions of the unary case only, the logits are not normalized with softmax, but just the indexes of the largest logit values are returned.

3.3 Evaluating the CNN

As a first step, we try to use the features and models produced by the author, and rerun their evaluation script. This is documented in their repository's README ("Evaluating the CNN").

A necessary prerequisite is extracting the CleanEval features. To that end install pip package 'click' (simple, but undocumented) and create your own Scala script using the snippets provided in the corresponding README section. Finally run `data/convert_scala_csv.py`, then rename `block_features.npy` to `cleaneval.npy` (undocumented).

Some sort of version difference necssitates adding the additional argument `allow_pickle=True` to function call `np.load()` in `data.py` . Further, there was an issue with interpretation of strings as bytes (or not) - possibly indicating some Python 2 versus Python 3 incompatibility. I fixed it by adding the following snippet to `data.py`:

```
for el in cleaneval:
    k = list(el.keys())
    kb = [bytes(key, 'utf-8') for key in k]
    for i,_ in enumerate(k):
      el[kb[i]] = el[k[i]]
```

Finally, we can run the evaluation script as described. However, take care: while changing the CHECKPOINT variable does change the model used, the evaluation is by always done on the CleanEval test-set. To change this, one needs to make code changes to the `test_structured` function in main.py: change all occurrences of variable `cleaneval_test` to the intended test-split, e.g. `web2text_test` (needs to be imported from data.py).

There is another minor inconcistency: they say in the paper that the original CleanEval train-split has 55 pages. However, in their code, they use 56 pages. Furthermore, it is notable that 796 webpages are present in the original pages, but in either split they use only 737.

3.3.1 Encoding problems of input files. When running the three steps of boilerplate removal, comprising of feature-extraction, label classification and the application of labels to a webpage, for a single html file, we noticed different encodings of the original html webpage input files. For example, when trying to run `ApplyLabelsToPage` with the `"orig/1.html"` as input file, one gets a `java.nio.charset.MalformedInputException` error, which prevents one from getting a result. We found out that the encodings of the input html files are different, and this error occurs because the file is "iso-8859-1" encoded. We wrote a shell script (see `to_utf8_linux.sh` or `to_utf8_mac.sh`) for converting input files to an utf-8 encoded file, which can be used when this error occurs. After conversion to utf-8 the html file could be processed successfully by the third step.

3.4 Training the CNN

Given the fixes we made for evaluating the CNN, training it worked without issue. Just like for evaluating, explicit code changes need to be made to use different splits. Replace occurrences of `cleaneval_train` and `cleaneval_validation` with the desired variables.

Model	Split	Acc.	Precision	Recall	F1
None (Reported)	CleanEval	0.84	0.88	0.85	0.86
Given Checkpoint	CleanEval	0.843	0.869	0.865	0.867
Trained	CleanEval	0.832	0.868	0.843	0.856
Trained	Ours(CE)	0.835	0.860	0.855	0.857
None (Reported)	Web2Text	0.86	0.87	0.9	0.88
Given Checkpoint	Web2Text	0.858	0.859	0.909	0.883
Trained	Web2Text	0.842	0.831	0.920	0.873
Trained	Ours(W2T)	0.875	0.889	0.910	0.890
Trained	Ours(New)	0.878	0.876	0.916	0.896

Table 1: Individual Run Results for the various configurations

3.5 Results

We now systematically compare results. In the paper, the original CleanEval split (56/5/676 train/test/split) and their own Web2Text split (531/58/148) of the CleanEval webpages are considered. For both, we report their numbers, evaluate their pretrained checkpoints (*Given Checkpoint*) and the model trained from scratch by us (*Trained*). In addition we create three new splits:

- *Ours(CE)*: a random split of the pages with the CleanEVal set sizes (56/6/676)
- *Ours(W2T)*: a random split of the pages with the Web2Text set sizes (531/58/148)
- *Ours(New)*: a random split of the pages with set sizes devised by us: 369/123/245. The intuition here is to take sizes that are in-between the CleanEval and the Web2Text split.

The new splits were implemented by adding new train test and validation objects analogously to the pre-existing ones in *data.py*. Then, as discussed previously, variables need to be replaced for training and evaluation (see the respective sections). Find the Results in Table 1.

The results are similar. The differences, however, are more than just rounding errors. Therefore, we investigate this further with statistical signifcance tests of multiple runs.

3.5.1 Significance testing. We wanted to check if the reported results can be reproduced. For this we ran the training and evaluation step 20 times to get 20 samples. All metrics of these 20 runs are then used for one-sample-t-tests, with the metrics reported in the paper as the baseline.

It is important to note that their splits used fixed sample IDs. This might influence the results by a lot. We therefore ran each split with their used samples and with random sample IDs as well. The splits of the random sample IDs had the same size as the other ones.

As we found a deviation in implementation specifics between the paper and the implementation provided in the repository, we clarify that: *we use the provided implementation here ($\lambda = 1$, no decay).*

We consider the following configurations:

- CleanEval: we use their fixed CleanEval split. See Table 2.
- Web2Text: we use their fixed Web2Text split. See Table 3.
- CleanEval-Random: we use their CleanEval split distribution, but choose the samples randomly every time. See Table 4.

Metric	t-statistic	p-value	mean	std	Paper
Accuracy	0.0075	0.9940	0.8400	0.0046	0.84
Precision	-4.1561	0.0005	0.8658	0.0148	0.88
Recall	3.6228	0.0019	0.8632	0.0159	0.85
F1	5.3142	0.0000	0.8643	0.0035	0.86

Table 2: Cleaneval (fixed) significance testing N=20

Metric	t-statistic	p-value	mean	std	Paper
Accuracy	-7.8660	2.1515e-07	0.8506	0.0052	0.86
Precision	-11.5793	4.7220e-10	0.8444	0.0098	0.87
Recall	10.0436	4.9065e-09	0.9168	0.0075	0.90
F1	-1.1066	0.2822	0.8791	0.0035	0.88

Table 3: Web2Text (fixed) significance testing N=20

Metric	t-statistic	p-value	mean	std	Paper
Accuracy	-1.0667	0.2994	0.8378	0.0088	0.84
Precision	-6.1021	7.2330e-06	0.8530	0.0197	0.88
Recall	3.5018	0.0023	0.8733	0.0298	0.85
F1	1.2883	0.2130	0.8625	0.0087	0.86

Table 4: Cleaneval (random) significance testing N=20

Metric	t-statistic	p-value	mean	std	Paper
Accuracy	-0.6861	0.50089	0.8575	0.0156	0.86
Precision	-3.0670	0.0063	0.8508	0.0279	0.87
Recall	1.9715	0.0633	0.9126	0.0287	0.90
F1	0.0208	0.9835	0.8800	0.0158	0.88

Table 5: Web2Text (random) significance testing N=20

- Web2Text-Random: we use their Web2Text split distribution, but choose the samples randomly every time. See Table 5.

For multiple configuration-metric configurations, we found significant differences. In a practical sense, the differences are not very major, however. In terms of being better than other baseline methods, we found that their statements hold.

3.5.2 Unary performance. The authors state the unary version, i.e. without the use of the pairwise potentials, is only marginally worse. We compare the means of our multiple runs in the fixed sample ids to check their statement. See Table 6 and Table 7. We find that the differences really are marginal.

3.5.3 Performance of adding L2 regularization and varying Lambda. We reported above that the code is inconsistent with the claims in the paper regarding L_2 weight decay and λ. We also trained and evaluated the CNNs with the settings from the paper, that is L_2 regularization with weight decay rate of 10^{-4} and interpolation factor $\lambda = 0.1$.

Metric	structured mean	unary mean
Accuracy	0.8400	0.8372
Precision	0.8658	0.8654
Recall	0.8632	0.8595
F1	0.8643	0.8617

Table 6: Cleaneval (unary) N=20

Metric	structured mean	unary mean
Accuracy	0.8506	0.8507
Precision	0.8444	0.8456
Recall	0.9168	0.9152
F1	0.8791	0.8789

Table 7: Web2Text (unary) N=20

For that setting and the fixed Web2Text split the accuracy, precision, recall were significantly different (p-values of 0.00, 0.00, 0.00, 0.32), but not the f1 score, and the mean and std. were also comparable with the results in Table 3. When using the random Web2Text split, we have not found significant different results compared to the reference in the paper (p-value of accuracy 0.63, precision 0.45, recall 0.16 and f1 0.64), which is different to our previous results in Table 5.

Our results on the CleanEval fixed split, showed significant differences to the reference in the paper with p-values for accuracy of 0.02, precision of 0.0, recall of 0.0 and f1 of 0.0, which is similar to our previous run results in Table 2 where we noticed significant differences for all metrics except accuracy.

When using the CleanEval random split for this setting, we also noticed different results compared to those reported previously in Table 4. We found significant differences for the accuracy and precision (p-value of 0.02 and 0.00 respectively), but no significant difference for recall and f1 (p-value of 0.12 and 0.42 respectively), whereas the results in the table were only significantly different for precision and recall.

3.6 Conclusion

Trying to reproduce this paper, we encountered a number of difficulties. Running the code required careful consideration of the packages and versions used. And even still, code changes were necessary for succcessful execution. Furthermore, there were a number of inconsistencies between the code and the paper.

In our evaluation of the model, we found similar, but different results. In a number of configurations, even statistically significant differences. Importantly, however, they still outperform the baselines, like they stated in the paper. Also, their statement about a marginal performance difference between the full and the unary version, is found confirmed.