Text classifier coding challenge

In this challenge I had to create a text region classifier using outputs from hypatos OCR engine. The code developed during this challenge can be found in the github repo
`https://github.com/EnriqueSMarquez/TextClassifierChallenge`.

The code is divided into scripts and notebooks. In this case I decided to perform evaluations and preliminar analysis of the data using jupyter notebook. In addition, to train any of both models one can use the `train.py` and data processors can be found at `data_procesing.py`. Sample training terminal command:

Transformer: `python train.py --datapath ./data/hypatos-ds-train.json --saving-path ./run13-selfattention-bbox-encoding --verbose 2 --model selfattention --feature-compression raw --min-repetitions 10 --batch-size 2`
BOW+SVM: `python train.py --datapath ./data/hypatos-ds-train.json --saving-path ../run1-bow-word-svm --verbose 2 --model svm --feature-compression bow --min-repetitions 10 --batch-size 2 --dataset-chopped-portion 0.85`

Since there was limited amount of time for this challenge I decided to first try a simple non-deep learning model. Hence, I implemented a system which uses kmeans (`K = 128`) to find "distance to cluster representation" of every word. To speed things up I concatenated one-hot encoded words along with the one-hot encoded discretisied coordinates (including page). As a result, the dataset drastically decreases in size (from (`num_words,dictionary_size`) to (`num_words,K`). We then proceeded to train an Support Vector Machine (SVM) using the gathered feature vectors. This model was trained at a word level, which makes the model oblivious to temporal relationships between entities.

For the second approach I decided to use a transformer and modified the text translator into a text classifier. In order to achieve this, I removed the decoder and alone work with the encoder. We then connected a dimension-wise fully connected layer at the output of the encoder to compress the output matrix into a vector with the same length as the input sequence. This allow us to have a transformer classifier with symmetric input and output lengths. One could connect a standard fully connected layer at the last stage of the network to perform the classification, however, the model complexity would increase as well as the number of output units. We directly target the region indexes and aim to predict which regions are entities. This model takes under consid-

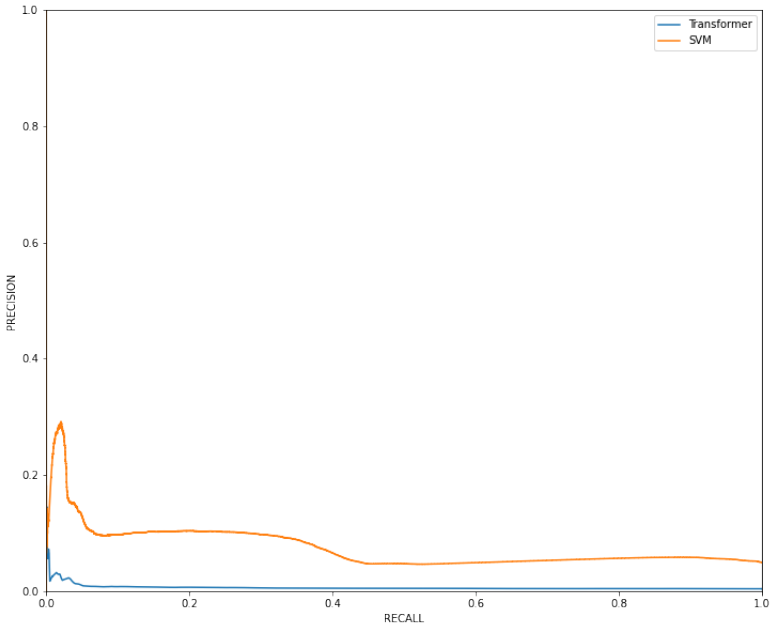
eration the relationship between words and their position on the document grid. The `TransformerTextClassifier` can use the traditional positional encoding (using sine and cosine waves with fixed positions), this would allow the model to learn temporal relationships from the OCR output order. In addition, I implemented a positional (one-hot) encoding which includes XY coordinates of the centroids for each word. The width and height information of every box was discarded and could be included to further improve the model. Note: The transformer architecture was taken from the original repository (`https://github.com/jadore801120/attention-is-all-you-need-pytorch`).

Dataset description

The dataset contains a dictionary of 47483 words when excluding words repeated less than 10 times. The majority of of the words are located on page 1, however, documents might contain up to 8 pages. Approximately 25% of the training words are assigned to the tag 'OTHER'. Some quick analysis on the dataset can be found on the notebook `data-misc-analisys.ipynb`.

Results

For this particular problem the metric should be dependant on the system needs. Typically I would compare models by analysing PR curves. If the system needs a high recall or precision, the PR curve can be used to select the threshold that matches the desire recall and precision. One could also directly compare models by using the area under the PR curve. Accuracy is not a robust performance metric since it only accounts for recall. Weighted F1 score could also be used as good metric for model performance.



As we can se on the PR curves, none of both models learnt a meaningful generalised rule to identify entities. Given my personal limited resources and time I could not debug the network or data processing any further. Nevertheless, by executing error analysis on validation and test set one can spot the discrepancies in the network and drastically improve its performance.

When having to look into errors at a higher resolution, it is worth using metrics such as: recall vs bbox width & height & position, precision vs bbox width & height & position. This will give us an insight of were the errors are happening in the model and why. However, at this point of the models, recall and precision curves are sufficient to make a comparison.

The notebook `models_evaluation.ipynb` includes the evaluation of the models. (Not outstanding) Weights for both models are added to the repo and can be found in **run1-bow-word-svm** and **run13-bow-word-svm**

Limitations and further approaches

The important next step is to perform error analysis and debug the pipeline. Both models can be drastically improved by adding the width and height information of the bounding box. Both models can also benefit from hyperparameter tuning.

Transformer specific improvements. When visualizing the errors that the transformer was making, I could see that many of them were ”loner”

predictions, which is not usually the case for these entitites. Hence, pospro-cessing ca further help remove those false positives. I believe the positional information might be lost in the network, hence, the positional encoding for this model might be better fit at the end of the decoder followed up by some learnable units. Using the non-encripted words would enable us to use pretrained embeddings and obtain more meaningful representations of the words. At the moment the model learns its own embedding. Something worth trying is changing the output to targe start and end tokens (corre-sponding to start and end entity) rather than regions indexes.