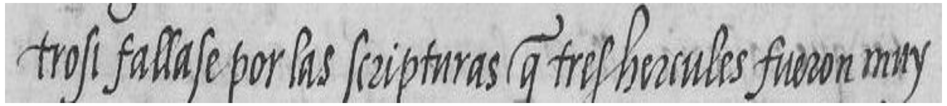
AI of GOD 3.0

OCR based LLM to recognize text from old Manuscripts

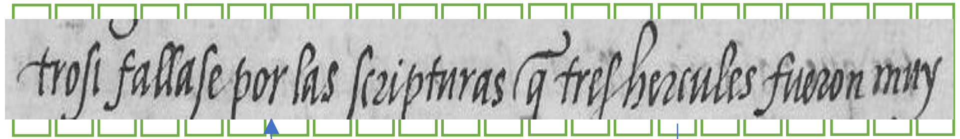
Team Name : Aditya Raj
Team Member : Aditya Raj

Methodology, Preprocessing and Architecture



Text Description of Image

Image Patches



TrOCR/Base-Handwritten

Text Description of Image

1. Finetuned on the training dataset to work for Spanish text.

2. Model was trained for 5 epochs and split into 80% train- 20% test for the original train set.

A t5 based text-text grammar correction model was used, the train set ‘transcription’ was used as correct-grammar and algorithm(‘transcription’) was used as wrong-grammar.

T5/base-English-grammar was finetuned on this Spanish dataset created after passing it through an algorithm, as Spanish and English are both derived from Latin, we can be sure that even not much train data will equip the model with very good capabilities.

Inference

Algorithm for creating the dataset for grammar-correction

Step 1: Normalization on Train Dataset

- Applied normalization to the training set text:
- • Replace 'ç' → 'z'
- • Remove accents with unidecode
- • Replace 'ñ' → 'n'
- • Handle 'q' → 'que' with cap symbol
- • “rr” → “r”
- • “ss” → “s”

Step 1: Character Removal and Modifications

- Removed 2% random characters and 3% of specific letters (a, o, e, i, s, n)
- Interchanged and removed (u, v) and (f, s) randomly
- Randomly split words with '_', ':', '-'
- Interchange (f,s) in 0.5% of occurrences
- Interchange (u,v) in 0.3% of occurrences

A dataset of top-100,000 words was downloaded, and characters were arranged in the order of maximum occurring order. Using this we removed a, o, e, i, n as the probability that these words come more often increases.

Word Length	Letters Used
2-letters	a, o, e, i, s, n, u, d, m, v, p, c, r, l, y, t, h, x, b, g
8-letters	a, e, o, i, r, s, n, c, t, d, l, u, m, p, b, g, v, f, h, j
14-letters	e, i, n, a, o, c, t, s, r, m, d, p, l, v, u, f, b, g, z, x

```
For this we need a dataset of bad Spanish
/ good Spanish, for this we used the
training set text with modifications,

0. The model output test_predict.csv was
checked and it detected 'n-accent' as
'A_', wrote a code to replace all 'A_' by
'n'.

1. Removed around 2% of characters, as
our model was giving WER around 0.27 at
this point of time, so removing 10% =
2.7% of random characters, also remove 3%
occurrence of 'a', 'e', 'n' in
particular.

-- This will take into account for the
missing words, 'z', 'n-accent', 'q' and
others.

2. Interchanging {(u,v), (f,s)}, removing
u, removing v, removing both at random,
do the same for f and s. This injection
into the code is done at random, and it
is only done for 50% of the time.

**More changes to lower WER, and to get
better results**

3. (f,s) were interchanged in some of the
places they occurred at random and (u, v)
at lesser than (f, s). change:
(f,s):0.5%, (u,v):0.3%

4. Around 3% of the words at random were
split from between with a '_' or ':' or
'-'.
```

Loss Function

Cross-Entropy Loss

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

M = 110, as there are 110 different characters so, we have a classification problem of 110 labels.

Metrics Observed

WER (Word Error Rate)

WER=(S+D+I)/N

Then we calculated the average WER over all iterations.