Asignación automática de dominios a definiciones

Josu Barrutia
Estudiante / Facultad de Informática EHU
jbarrutia006@ikasle.ehu.eus

1 Introduction

La asignación de dominios a definiciones es un problema de clasificación multiclase de texto. En este caso, se trata de entrenar varios modelos de lenguaje preentrenados para la asignación automática de dominios a conceptos de Wordnet. Concretamente, se utiliza la glosa y los posteriores ejemplos de un synset para clasificarlo en un conjunto de etiquetas llamado BabelDomains (Camacho-Collados and Navigli, 2017). Se llevará a cabo una búsqueda de hiperparámetros para conseguir realizar un fine-tunning de los modelos para lograr el mayor desempeño en la tarea sobre el conjunto de test. Se compararán varios modelos entrenados en esta tarea concreta con un modelo zero-shot Sainz and Rigau (2021) y con un modelo que haya sido entrenado con muy pocas instancias de la tarea en cuestión (few-shot).

2 Datos

Partimos de dos ficheros (wordnet_dataset_gold.txt y babeldomains_wordnet.txt) que contiene un gran número de synsets de wordNet y su respectivo dominio anotado. El fichero wordnet_dataset_gold.txt será usado tan solo para test, mientras que el fichero babeldomains_wordnet.txt será usado tanto para train, como para development, éste último fichero incluye unos índices de fiabilidad en la asignación de dominios a cada synset y, para determinados synsets se han clasificado en varios dominios distintos. Para arreglar estos problemas y limpiar los datos, nos quedaremos únicamente con el primer dominio con el que se clasifica el synset en cuestión y aplicaremos un filtro para quedarnos solo con las tuplas (synset,dominio) que presenten una fiabilidad del 100%. Tras arreglar unos pequeños fallos en el fichero, podemos aplicar la siguiente expresión de awk para quedarnos únicamente con las instancias que verifiquen las condiciones anteriores:

```
awk -F'\t' '$3 == 1 {print $1 "\t" $2}' \
babeldomains_wordnet.txt > babeldomains_wordnet.txt
```

Esto nos deja con 11927 i

Tenemos que eliminar los datos del fichero babeldomains_wordnet.txt que se repitan en wordnet_dataset_gold.txt, pues la partición de test tiene que tener instancias nunca vistas por el modelo. Para ello, aplicamos la siguiente expresión de awk:

```
awk 'NR==FNR{a[$1]; next} !($1 in a)' \
wordnet_dataset_gold.txt babeldomains_wordnet.txt \
> babeldomains_wordnet.txt
```

Esto nos deja con 11849 instancias para train y dev, y 1540 para test (wordnet 3.0 tiene 117659) y ya tenemos ambos ficheros limpios. Ahora tenemos que mapear cada synset con su respectiva glosa. Para ello, vamos a utilizar los ficheros de wordnet. En primer lugar, concatenamos todos en un único fichero. A continuación, nos quedaremos con el PoS+offset (clave única) y con su respectiva glosa. Los ficheros descargados tiene un Part of Speech que no incluye el fichero babeldomains_wordnet.txt, pues está clasificando algunos adjetivos como *satellite adjetives* con la marca "s", esto hace que algunos synsets no casen. Así pues, vamos a reemplazar todos los synsets "s" por "a".

```
cat data.adj data.adv data.noun data.verb > wordnet.data
awk -v OFS="\t" '{print $3$1,substr($0,index($0,"|")+2)}'
    ↪ wordnet.data > wordnet.data
sed 's/^s/a/' wordnet.data > wordnet.data
```

Y por último mapeamos cada synset con su glosa, para conseguir train.tsv y test.tsv, dos ficheros que contienen las tuplas (glosa, dominio):

```
awk -v OFS="\t" 'NR==FNR{map[$1]=substr($0,length(
    ↪ $1)+2); next} {$1=($1 in map)? map[$1] : $1;
    ↪ print}' wordnet.data babeldomains_wordnet.txt >
    ↪ train.tsv
awk -v OFS="\t" 'NR==FNR{map[$1]=substr($0,length(
    ↪ $1)+2); next} {$1=($1 in map)? map[$1] : $1;
    ↪ print}' wordnet.data wordnet_dataset_gold.txt >
    ↪ test.tsv
```

Una vez empezamos a cargar los datos con pandas, nos damos cuenta del mal balanceamiento que tienen las clases tanto en train como en test:

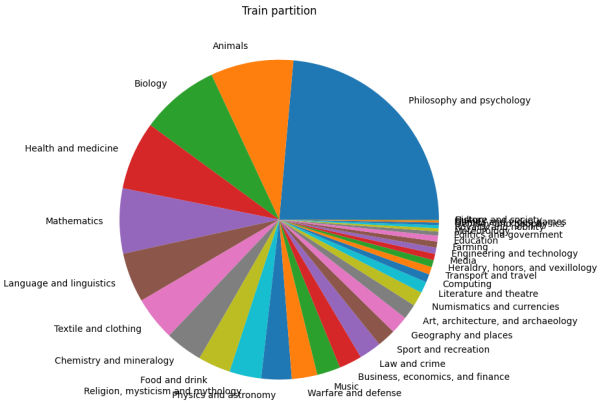


Figure 1: Distribución clases en partición entrenamiento

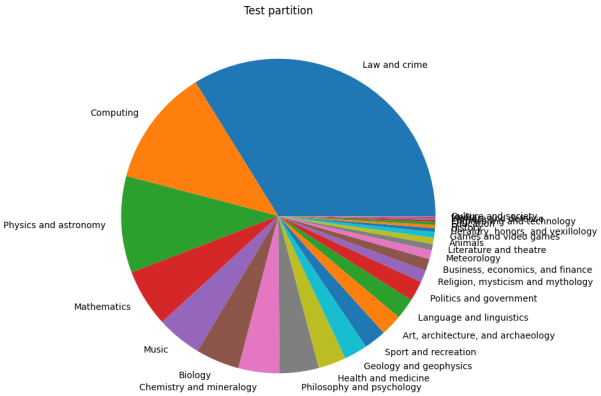


Figure 2: Distribución clases en partición test

3 Sistema

La mejor opción para realizar esta tarea de clasificación multiclase de texto es la arquitectura transformers. Seleccionaremos tres arquitecturas con sus respectivos checkpoints para el tokenizer y el modelo: distilbert-base-uncased (Sanh et al., 2019), bert-base-uncased (Devlin et al., 2018) y roberta-base (Liu et al., 2019). Los modelos serán entrenados y evaluados con NVIDIA GeForce RTX 3060 Mobile / Max-Q.

4 Experimentación

Para encontrar los hiperparámetros con los que consigamos un mayor acierto en test, tenemos que evaluar el modelo que estamos entrenando en la partición train sobre una partición disjunta que no sea la de test. Para ello, tenemos la partición de validación.

En primer lugar, queremos saber un valor adeacuado para el learning rate. Sun et al. (2019) señala que la tasa de aprendizaje es clave para evitar

el "Catastrophic Forgetting" (Olvido Catastrófico), donde el conocimiento sobre el lenguaje del modelo preentrenado se borra durante el aprendizaje de nuevos conocimientos. Tasas de aprendizaje menores que 4e-4 son necesarias para evitar el olvido catastrófico. Con una tasa de aprendizaje demasiado agresiva como 8e-4 el conjunto la pérdida en el conjunto de train no converge (ver Figura 3c. Ésta puede ser la razón por la que el paper original (Devlin et al., 2018) utilizará unas tasas de aprendizaje entre 5e-5, 4e-5, 3e-5 y 2e-5.

El tamaño del batch para el dataloader de entrenamiento adecuado dependerá de la tarea. Devlin et al. (2018) utiliza un tamaño de batch de 32 aunque indican que un tamaño muy grande de las secuencias de entrada (las glosas en este caso) puede hacer que el batch no quepa en la memoria de la GPU. En ese caso, habría que reducirlo a la mitad. En nuestro caso no tendremos problemas de memoria pues nuestras secuencias tienen una longitud bastante pequeña (ver Figura 4)

La variación del tamaño del batch no afecta en la convergencia de la pérdida del modelo en los conjuntos de entrenamiento y validación. No obstante, se puede ver como para un tamaño de batch menor la curva de la pérdida no es tan suave como en un tamaño mayor de batch. Esto ocurre porque cada iteración el descenso de gradiente y el ajuste de pesos se efectúa para menos instancias, por lo que el ajuste de parámetros es más preciso. Por lo tanto, para el fine-tunning final de los modelos utilizaremos un tamaño de batch del dataloader de entrenamiento de 32 instancias.

El número de épocas que necesita el modelo para aprender la tarea de clasificación es bastante pequeño. Devlin et al. (2018) indica que el número de épocas requeridas era muy pequeño, p.ej 3 épocas para las tareas GLUE. Al analizar la Figura 3a podemos ver como, a partir de la época 4, la pérdida en la partición de validación no disminuye, e incluso aumenta, mientras que la pérdida en entrenamiento mejora levemente. Esto nos sugiere que el modelo está empezando a sobreajustar demasiado sobre los datos de entrenamiento y estamos perdiendo capacidad de generalizar. Por ello, es una buena idea implementar la idea de Early Stopping o entrenar tan solo durante 4 épocas.

A las arquitecturas de los modelos preentrenados que se utilizan se les añade un feed-forward para convertir los 768 outputs en el número de dominios posibles que tiene nuestra tarea de clasificación. Pueden añadirse más de una capa feed-forward aunque con una es suficiente. Todos los parámetros del modelo BERT serán fine-tuneados, pero se puede congelar el modelo base y añadir más capas de clasificación sobre el modelo base BERT.

En cuanto al optimizador, el artículo original (Devlin et al., 2018) también utilizaba Adam con weight decay. Seguir usando el mismo optimizador sería lo más lógico, aunque se pueden probar otros diferentes. Nosotros utilizaremos AdamW (PyTorch).

References

Jose Camacho-Collados and Roberto Navigli. 2017. BabelDomains: Large-scale domain labeling of lexical resources. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 223–228, Valencia, Spain. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

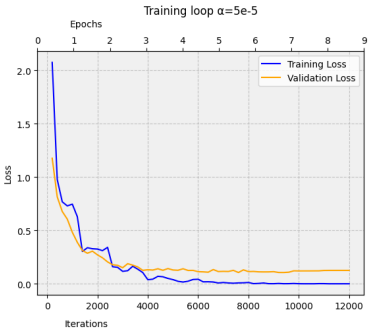
Oscar Sainz and German Rigau. 2021. Ask2Transformers: Zero-shot domain labelling with pretrained language models. In *Proceedings of the 11th Global Wordnet Conference*, pages 44–52, University of South Africa (UNISA). Global Wordnet Association.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

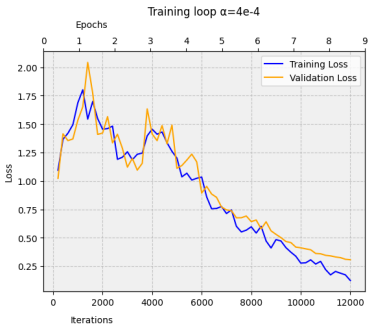
Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583.

A Appendices

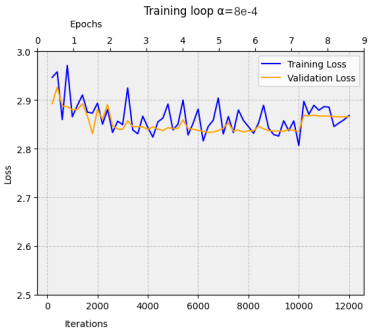
Appendices are material that can be read, and include lemmas, formulas, proofs, and tables that are not critical to the reading and understanding of the paper. Appendices should be **uploaded as supplementary material** when submitting the paper for review. Upon acceptance, the appendices come after the references, as shown here.



(a) Entrenamiento tasa adecuada



(b) Entrenamiento tasa alta



(c) Entrenamiento con olvido por tasa alta

Figure 3: Comparación entrenamiento con distintas tasas de aprendizaje

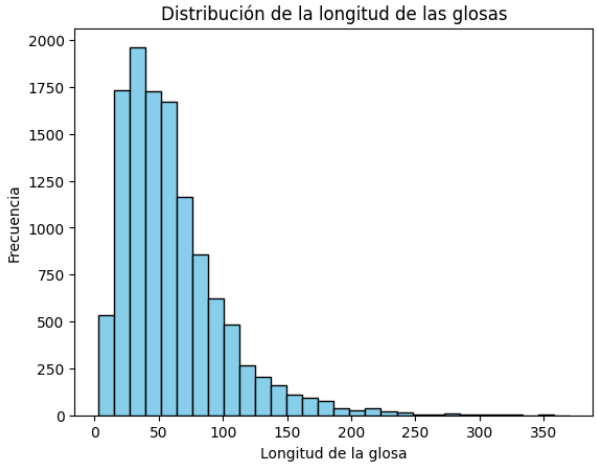


Figure 4: Distribución de la longitud de las secuencias

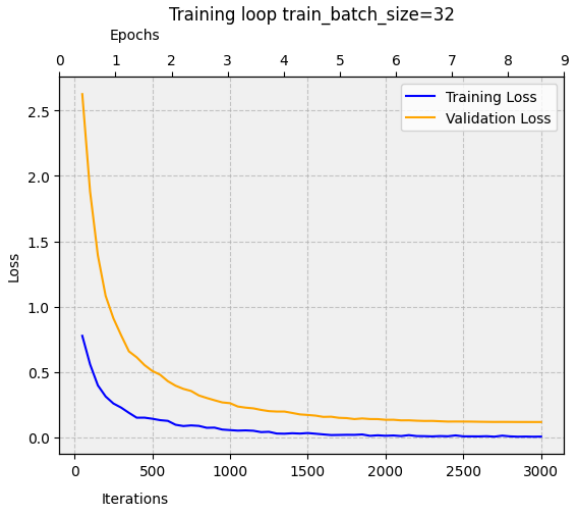
LaTeX-specific details: Use `\appendix` before any appendix section to switch the section numbering over to letters.

B Supplemental Material

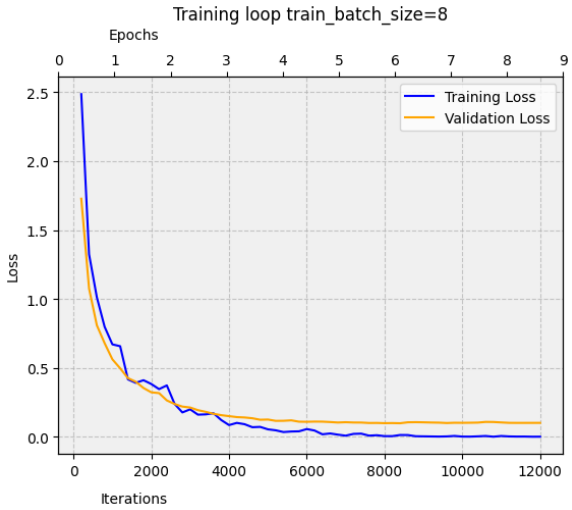
Submissions may include non-readable supplementary material used in the work and described in the paper. Any accompanying software and/or data should include licenses and documentation of research review as appropriate. Supplementary material may report preprocessing decisions, model parameters, and other details necessary for the replication of the experiments reported in the paper. Seemingly small preprocessing decisions can sometimes make a large difference in performance, so it is crucial to record such decisions to precisely characterize state-of-the-art methods.

Nonetheless, supplementary material should be supplementary (rather than central) to the paper. **Submissions that misuse the supplementary material may be rejected without review.** Supplementary material may include explanations or details of proofs or derivations that do not fit into the paper, lists of features or feature templates, sample inputs and outputs for a system, pseudo-code or source code, and data. (Source code and data should be separate uploads, rather than part of the paper).

The paper should not rely on the supplementary material: while the paper may refer to and cite the supplementary material and the supplementary material will be available to the reviewers, they will not be asked to review the supplementary material.



(a) Entrenamiento con tamaño 32



(b) Entrenamiento con tamaño 8

Figure 5: Comparación entrenamiento con distintas tamaños de batch