Paweł Paściak, 24.01.2024

Furniture Stores

explanations for the ML Engineer Assignment

My project pipeline and conclusions

1. I did some preliminary data analysis. I scrutynized more than a dozen working sites from the attached list, reviewed their HTML code, verified that a good portion of them are prepared with Shopify tools, wondered whether Shopify API could be used, but gave up these attempts in favor of the universality of the solution, supposing also that access to the API may involve various limitations.

2. I noticed that most often product names are associated with certain HTML tags, especially those with the *class* attribute, the value of which is a string of which the word *'product'* is a part. I experimented a lot with other terms, and wondered whether to also include links to other pages whose parameters also include the word 'product'.

3. Finally, I wrote a tool called Tagger, the part of which is a prototype of a parser of HTML search logic. The task of the tagger is to build an annotated dataset, which can then be used to build a model that captures product names in the text of the page itself, without reference to its source code.

Figures in data preparation:

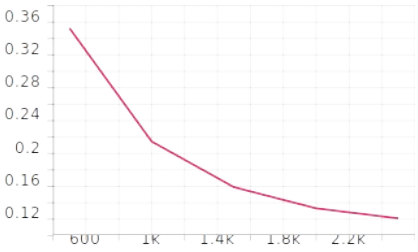
pages in the file attached to the task	704
active sites	<500
pages containing products	127
number of found products on all pages	518
number of sample phrases in the training set	2524
number of words to classify in the training set	16023
number of sample phrases in the test set	37
number of words to be classified in the test set	222

4. I subjected the result of the tagger to manual cleaning so that it could be used in training the model (the problem was mainly punctuation marks coming into collision with list element separators). To increase reliability, I separated the test set by reviewing the examples creating it; this way I was sure that the model evaluation would evaluate the effects I wanted to evaluate. I then balanced the numbers of examples making up the two sets, so that the numbers of "product" and “non-product” examples were of the same order of magnitude (there were also small fractions of mixed examples in both sets).

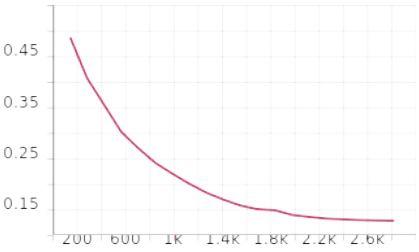
5. I then finetuned the model using the standard procedure for named entity recognition. I tried several values for learning rate and batch size, but I considered broader tuning of hyperparameters to be beyond the scope of activities at this stage. I evaluated the model using a standard set of metrics - the best result is:

Train Loss	Valid Loss	Precision	Recall	F1	Accuracy
0.1332	0.1320	0.7609	0.8974	0.8235	0.9691

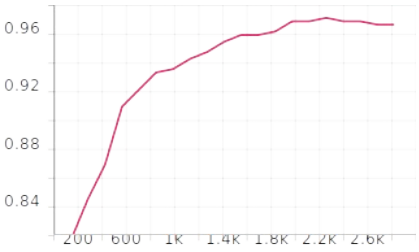
Based on the training loss and validation loss graphs attached in the charts, it can be concluded that despite the relatively small training set, the model is not overtrained. The accuracy and F1 values obtained indicate that the approach taken to the problem provides a promising basis for further work.



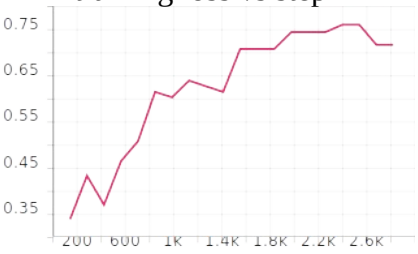
training loss vs step



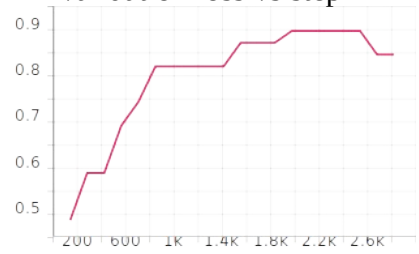
validation loss vs step



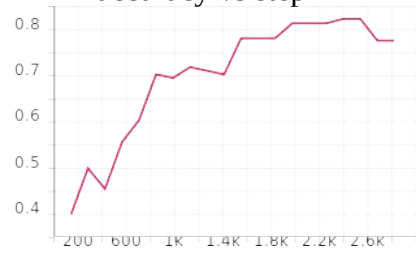
accuracy vs step



precision vs step

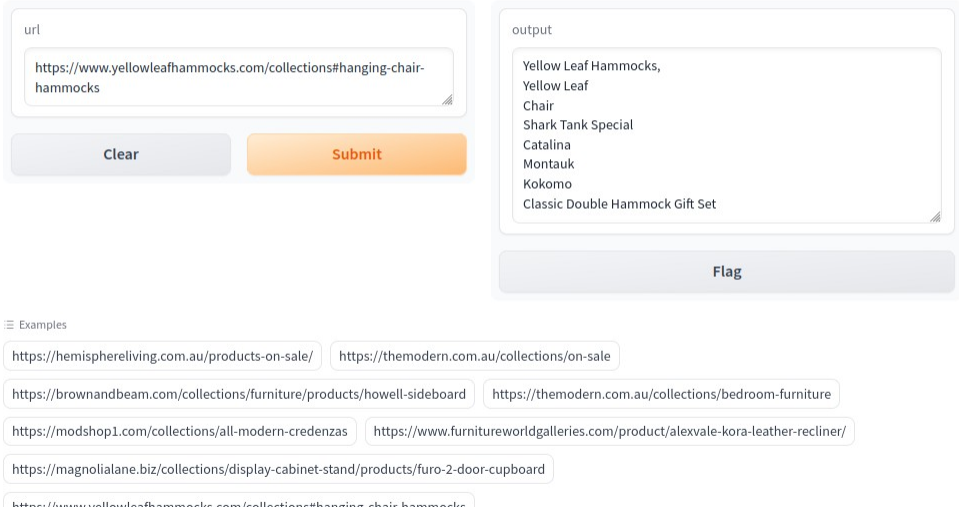


recall vs step



F1 vs step

6. Finally, I create a simple application using the Gradio framework to conveniently test the model in sanity check mode. The examples for which the results are cached are chosen to present the strengths and weaknesses of the solution as objectively as possible.



Assumptions made about the task:

- the search for products applies only to the specified pages, without going to other pages of the site
- if the page redirects to another page, I follow the redirect and search on the landing page
- I take a broad definition of "product" - also non-furniture, gift certificates, loyalty programs
- The target solution should search not only for products in lists, but also those that are elements of longer texts
- also, we consider words like "Chair" or "Wardrobe" as products because there are indeed such non-specific names, although the correct classification will depend on the context

Coding style assumptions:

- I write universal code, so that it is easy to change the subject or purpose of the analysis
- I write modular code, separating the different phases of work on the solution (acquisition and preparation of data sets, training, inference), separating the handling of the definition of the search logic (tagger.ini file) from the implementation of the search (tagger.py).
- I do not mimic production code, but write code that can be easily rewritten into production code - for example, functions that perform specific tasks (even in Jupyter Notebook, I use a style that can be easily rewritten into a reusable program).

ML assumptions:

- I establish a baseline for possible follow-up: I run a standard procedure using the most common version of the Bert model, avoiding intensive parameter tuning
- I use a permanent separation between training and test collection to avoid the possibility of data leakage when the procedure is repeated many times and to give confidence in a reliable test collection
- as in the base model's tokenizer, I use truncation of examples to 512 tokens
- optimize the model using the F1 metric

Possible extensions / improvements / future work directions:

- use of extreme gradient boosting (XGBoost, LightGBM) to better classify product-non/product content during data preparation
- use of Scrapy type crowlers to massively increase the number of examples
- adding the ability to analyze elements dynamically loaded by javascript (at least during inference)
- augmentation of the training set - insertion of product names in various specially designed contexts
- relying on Shopify's API to accurately search product names

Known problems:

- insufficient support for product names split by tokenizer
- only one base model considered - I should test at least one more
- the collection produced by the tagger needs to be cleaned up (which I did manually) - a better solution would be to validate and correct the format within the tagger

Technologies used:

Python v. 3.8	whole solution
PyCharm	IDE
Jupyter Notebook	train/test set preparation, parameters setting, training, evaluation
BeautifulSoup	websites scarping
bash/vim	data cleaning + train/test set split
Pandas	some data processing
Hugging Face Transformers Hugging Face Datasets (with Pytorch in background)	training pipeline
TensorBoard	monitoring of metrics
Gradio	inference app preparation
HF Spaces	inference app deployment
HF Models	model deployment
github	version control, code sharing