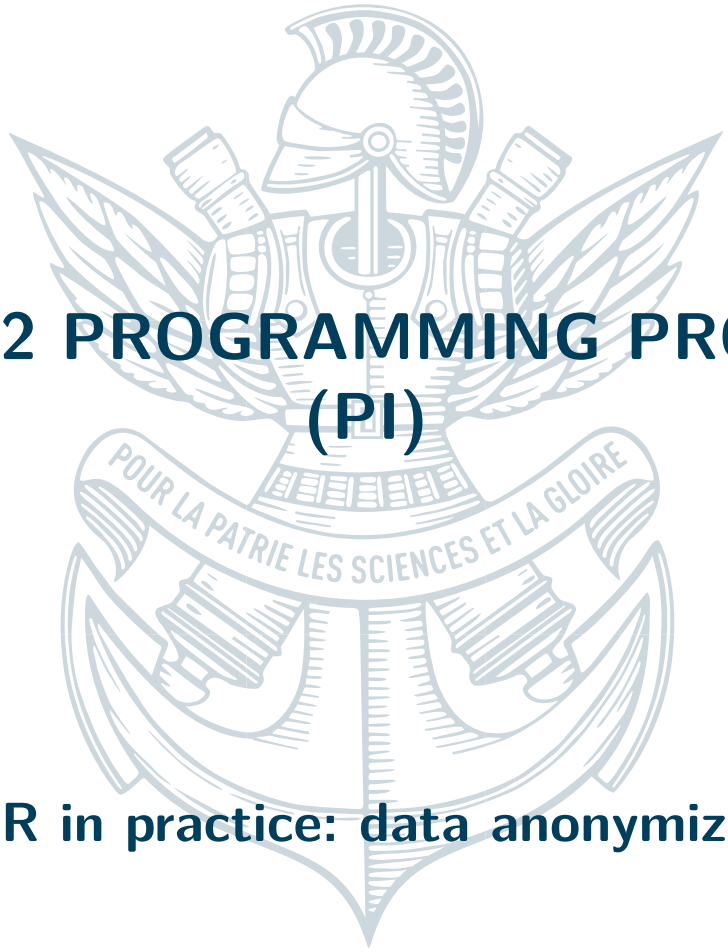

INF 442 PROGRAMMING PROJECT (PI)

GDPR in practice: data anonymization

18 May 2020

Duc Thinh NGO,Chin Wei CHONG

1 SUB-PROBLEM 1

The objective of this sub-problem is to adapt our algorithm constructed during the TD to the training and testing files with format .npy, which is constructed for the purpose of reducing size of the data files, since .csv files are extremely large. To achieve our goal, we use a library to load the data from .npy files, namely libnpy [1]. This library allow us to define some methods in the dataset to load the data from the .npy files. As a result, we can use what we have done previously to tackle the problem. However, the library has not yet provided the methods to read the unicode data stored in .npy format. Thus, we had to reproduce the true_labels files under the integer data type.

The following image illustrates the results of the test_knn with only 150000 samples in training files and 10000 samples in testing files (Here we are tackling the data ConLL-2003 [2], as there are too many data - 203593 instances, it takes very long time to generate the exact result, we look at the problem in a smaller scale to ensure the algorithm works) :

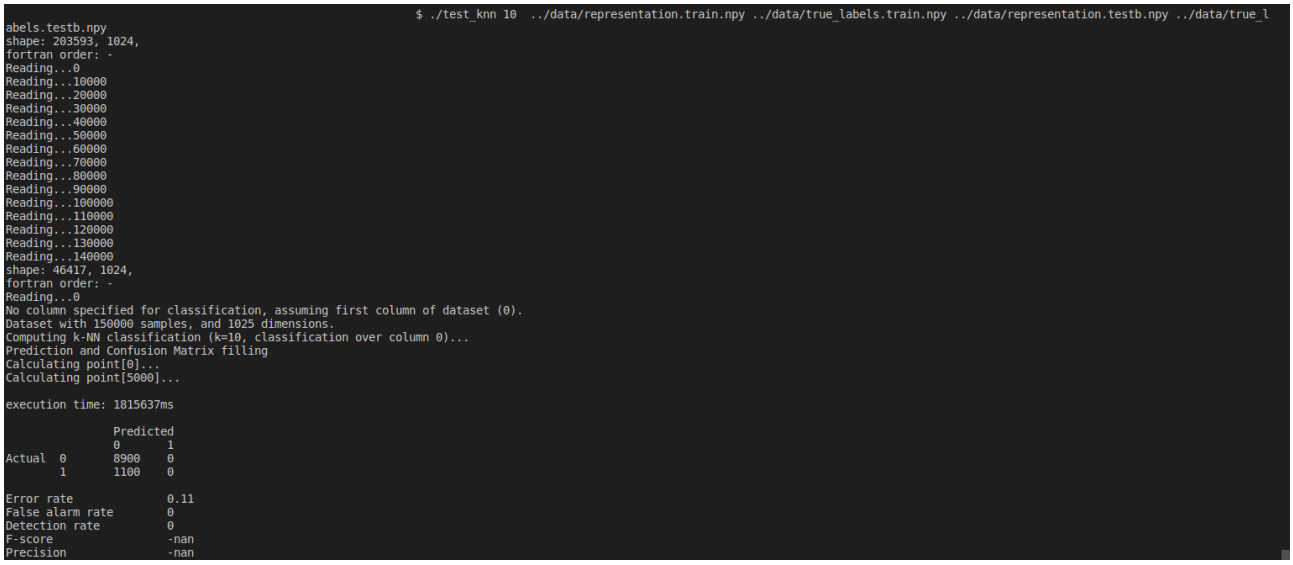

Figure 1 – The associated running time and the confusion matrix of k-NN with k = 10

Of course the results can be greatly improved if we increase the number of samples !

2 SUB-PROBLEM 2

We make use of the notebook provided to preprocess the data with a pretrained XLnet [3]. First published in 2019, the method stays as the state-of-the-art in NLP and has been announced to outperform BERT [4] on several tasks. Therefore, we can implement a pretrained XLnet on the dataset ConLL-2003 and the dataset WNUT17 [5] (in sub-problem 4) to compare their performance. Thanks to the pipeline proposed in the notebook, we can easily implement XLnet with only some modifications. Here, we implement on Google Colab with a free GPU enabled to improve the time computing. Besides, we detected a problem in the provided notebook with the RAM memory. In particular, a numpy array was used to store the vectors calculated by the model (BERT/XLnet). Despite the advantage of memory and time computing efficient, it is required to make a copy every time we append a numpy array to another one, which is the main cause of the running out of the RAM memory. Remark here that the problem still remains even when the notebook runs on Google Colab with a 12GB memory of RAM. To resolve this problem, we used the native list of Python instead, which acts as a linked list and of course does not require to make a copy when we do append. At the end, we have succeeded to produce the representation of the dataset by using an XLnet. The performance of this auto-encoder is presented at the sub-problem 3 and 4.

3 SUB-PROBLEM 3

The classifier used earlier in the first sub problem is k-NN classifier, which is one of the zero-one classifiers. For such binary classification problem (classify person vs rest), we can of course analyse the data using other type of algorithms, for example, what we have implemented in C++ are linear regression for classification and logistic regression for classification (only binary). The idea is explained in the lecture 8, which is to determine the coefficient β from the training set and estimate the true labels of the testing data. These models are fitted by least square methods and they are differed by the discriminant functions used. As a result, the linear regression for classification gives us a bad result with a moderate execution speed whereby the logistic regression for classification gives us a satisfied result with very long execution time, since it needs convergence and easily influenced by the curse of dimensionality. To overcome this issue, we can implement random projection on these methods although it will slightly influence the results. We work on first 50000 samples of ConLL-2003 preprocessed by BERT.

The results are as follows (check next page for the table).

In fact, it is easy to note that since the class Others outnumbers the class Person, the k-NN model which relies much on a major voting may turn out to perform poorly and to predict almost as Others. Therefore, in the next part, we will use the original NER tags instead of

C++ program of ConLL-2003 preprocessed by BERT, random project to dim 10, Rademacher						
Method	Error	False alarm	Recall	Precision	F-score	Time (ms)
k-NN with k = 10	0.0597	0	0	nan	nan	35812
linear regression	0.0601	0.0005	0.0004	0.0455	0.0007	9162
logistic regression	0.0606	0.0010	0.0004	0.0227	0.0007	10652

binary ones. Moreover, the dataset cannot ensure to be linearly separable, the linear regression thus rarely performs well in this case. However, the logistic regression seems to be ideal in the theory, since it combines of a both linear and non-linear function, which could somehow fits better in a dataset in general. Despite that fact, the time for the logistic regression to converge is usually very long, specifically in our problem, as demonstrated. Note that even when we tried to implement the logistic regression on python with the sckit-learn, it still takes a lot of time for the convergence.

Therefore, we decided to make use of Google Colab with the advantage of a free GPU enabling us to save time on training. To make the best out of the GPU, we use the library PyTorch [6] and CUDA. In the implementation of logistic regression, instead of using LBFGS optimizer, we use here Adam, a common optimizer of type gradient descent. The output is a vector of 9 compositions, demonstrating the probability of each class. Thus, the loss function used here is a cross entropy loss. The training is repeated over 1500 iterations. With the help of CUDA, it takes only a minute to finish the training on the whole dataset and predicts the result right immediately. The performance is impressive and far better than k-NN.

(Note that macro-average compute independently for each class and then take the average (hence treating all classes equally), whereas a micro-average will aggregate the contributions of all classes to compute the average (which is preferable if there might be class imbalance)

k-NN (k=30, projected to dim 10)			
Time spent : 37s			
Type	Precision	Recall	F1-score
O	0.85	0.99	0.91
B-MISC	0.00	0.00	0.00
I-MISC	0.31	0.06	0.10
B-PER	0.00	0.00	0.00
I-PER	0.48	0.04	0.07
B-ORG	0.00	0.00	0.00
I-ORG	0.56	0.12	0.19
B-LOC	0.00	0.00	0.00
I-LOC	0.60	0.25	0.35
micro avg	0.84	0.84	0.84
macro avg	0.31	0.16	0.18

Logistic regression for classification			
Time spent : 0s			
Type	Precision	Recall	F1-score
O	0.97	0.99	0.98
B-MISC	0.00	0.00	0.00
I-MISC	0.73	0.63	0.68
B-PER	0.00	0.00	0.00
I-PER	0.93	0.82	0.87
B-ORG	0.00	0.00	0.00
I-ORG	0.82	0.75	0.78
B-LOC	0.00	0.00	0.00
I-LOC	0.84	0.75	0.79
micro avg	0.95	0.95	0.95
macro avg	0.48	0.44	0.46

So far, taking the advantage of PyTorch, we advance to build a simple fully connected network, which is a common classifier in several deep neural networks nowadays. A simple

network, with only a hidden layer of 128 nodes using a rectified linear unit activation and a log softmax at the output. The training takes as much time as the logistic regression (which is actually a network with no hidden layer and a sigmoid activation). The results seem equivalent but the errors in training proved that this network converge faster than the logistic regression.

Below is the benchmark for 4 approaches that perform the best. Since the objective of the project remains as anonymization, we only observe the precision, recall and F1-score for the class I-PER (since there is no B-PER in the test set). Moreover, due to the sensibility of the objective, we suggest to rather look at the F2-score, which means the recall is more important. Remark that though the fully connected network helps the model to saturate at a lower training error, the results remain nearly the same on the test set. The XLnet does not improve the test results. The explanation could be that due to the simplicity of the dataset, a common used, easy model as the combination of BERT and logistic regression is already enough to solve the problem. Another observation is that the combination of XLnet and logistic regression performs poorly compared to all the rest. In the next sub-problem, with a more complex dataset, we will see if XLNet or fully connected network could outperform the rest.

Benchmark on ConLL-2013				
Method	Precision	Recall	F1-score	F2-score
BERT + Logistic Regression	0.92	0.82	0.87	0.84
BERT + Fully connected network	0.91	0.88	0.89	0.89
XLnet + Logistic Regression	0.92	0.28	0.44	0.33
XLnet + Fully connected network	0.90	0.87	0.89	0.88

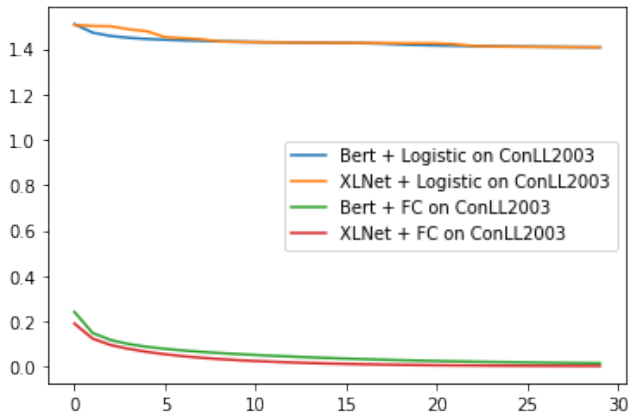

Figure 2 – Training errors over iterations on ConLL-2003

Just note that we have other tools as well to analyze the data given, for example the libsvm given in the TD environment, we can make use of libsvm-train to create a model and libsvm-predict to predict the data and analyze the accuracy. To use libsvm-train, we have also a program written in C to convert the .csv file into a format in which libsvm-train can read. However, the process takes very long time and we will not demonstrate the result here.

4
SUB-PROBLEM 4

Here we used another dataset for the testing, WNUT17 - a noisy user-generated dataset mined from several sources of social media as Twitter, Reddit, Youtube and StackExchange. This dataset is expected to be much harder than the ConLL2013 since it includes not only slang but also emoticons, hyperlinks and so on. Some examples from the training dataset :

— "I split my time now btwn SF and San Mete"

— "@Miss_SarahBaby hmmm...u kno when happy hour ends ?"

As expected, even when we implement XLnet/BERT with fully connected network as a classifier, the results remain weak compared to those on ConLL2013 which was mined from newspaper with a formal language. However, the fully connected network has proved to be more efficient than the logistic regression.

Benchmark on WNUT17		
Method	B-person F2-score	I-person F2-score
BERT + Logistic Regression	0.10	0.11
BERT + Fully connected network	0.49	0.48
XLnet + Logistic Regression	0.06	0.00
XLnet + Fully connected network	0.38	0.41

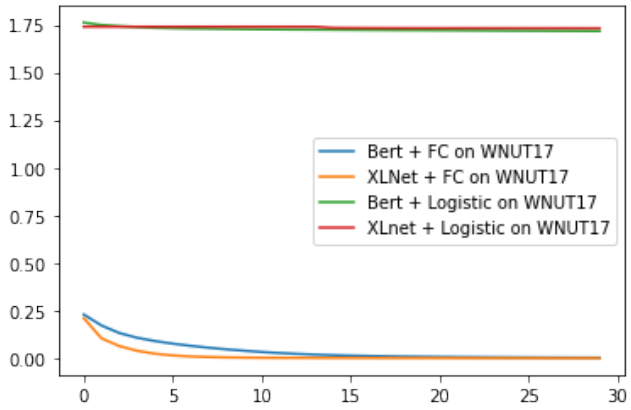

Figure 3 – Training errors over iterations on WNUT17

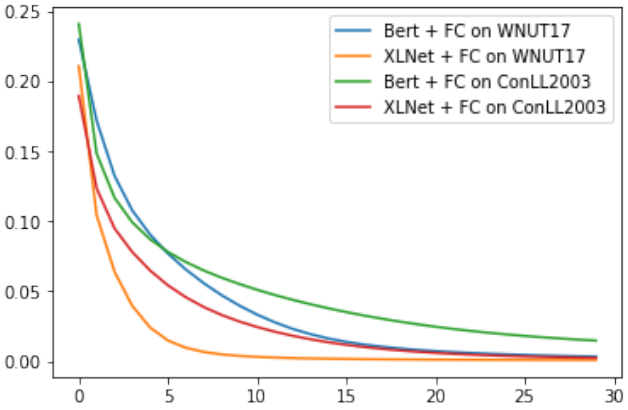

Figure 4 – Taking a closer look at BERT and XLnet

Once again, the XLnet remains to be outperformed by BERT though it allows the training errors to converge much faster. The reasons for that remain to be studied.

RÉFÉRENCES

[1] libnpy. `https://github.com/llohse/libnpy`.

[2] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task : Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, page 142–147, USA, 2003. Association for Computational Linguistics.

[3] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet : Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

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

[5] Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. Results of the WNUT2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch : An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.