DATA ANONYMIZATION PROJECT

GDPR in practise: Data anonymization

Coordinator : Adrien Ehrhardt

Michael Fotso-Fotso
Abderrahim Mama

Sommaire

Introduction

The scenario is that we are Data Scientists in an industry that deals with private data. We have transcripts from interactions people had with your product. Our team wants wants to sell this data to Amazonia, which will do some fancy machine learning to predict your users' behaviour, e.g. what they are going to buy next. The (legal, not moral) problem is that our data comes from many countries, which don't have the same legislation regarding private data, and our users probably did not agree to give away their private data to Amazonia. Our mission is to anonymize all these texts to remove any personal data before handing them out to Amazonia.

Datasets

In the first subproblem, we will be using **CONLL2003** In the fourth subproblem, we will be testing our two Machine Learning models on other datasets namely : **BTC, WikiGold**.

— **CONLL2003** : is a named entity recognition dataset released as a part of CoNLL-2003 shared task : language-independent named entity recognition. The data consists of eight files covering two languages : English and German.

— **Broad Twitter Corpus (BTC)** : This is the Broad Twitter corpus, a dataset of tweets collected over stratified times, places and social uses. The goal is to represent a broad range of activities, giving a dataset more representative of the language used in this hardest of social media formats to process. Further, the BTC is annotated for named entities.

— **Wikigold** : wikigold.conll.txt is a manually annotated collection of Wikipedia text, presented in :

Dominic Balasuriya and Nicky Ringland and Joel Nothman and Tara Murphy and James R. Curran (2009), in Proceedings of the 2009 Workshop on The People's Web Meets NLP : Collaboratively Constructed Semantic Resources (http ://www.aclweb.org/anthology/W/W09/W09-3302).

1 Steps before classification

— **First**, we have transformed the raw data to a format suitable to our subsequent analysis. Example : the raw text has to be tokenized into “tuples”consisting of a word and a label.
The words are tokenized : for now, think of it as some fancy stemming or lemmatization, i.e. we separate the root of each word (“playing” becomes “play” and “ing” where usually represents a word separation into several tokens). Think of each token as an observation, or a row of a matrix, e.g. a flower in the famous iris dataset ;

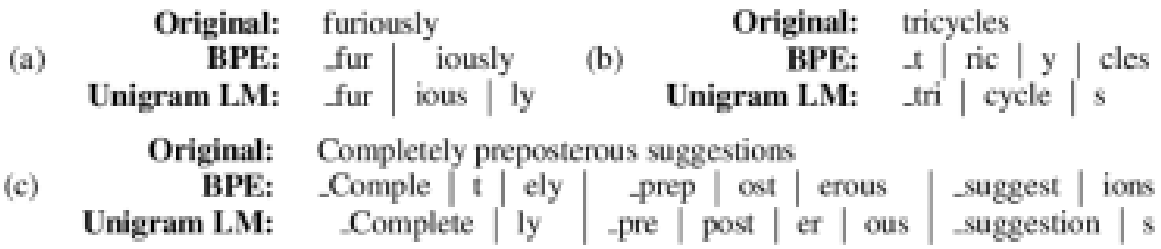

FIGURE 1.1: Tokenization

The previous step enables us to construct a dictionary : the list (of, say, length s) of all unique tokens in the document. Think of this dictionary as the Larousse where each word would be uniquely indexed.

— **Second**, Each token x is converted to a dummy vector which size s is the same as the dictionary, where a '1' indicates the presence of that token and '0' its absence, i.e. each token is now represented by a flipped "bit" at the location of its index in the dictionary and s 1 0s.

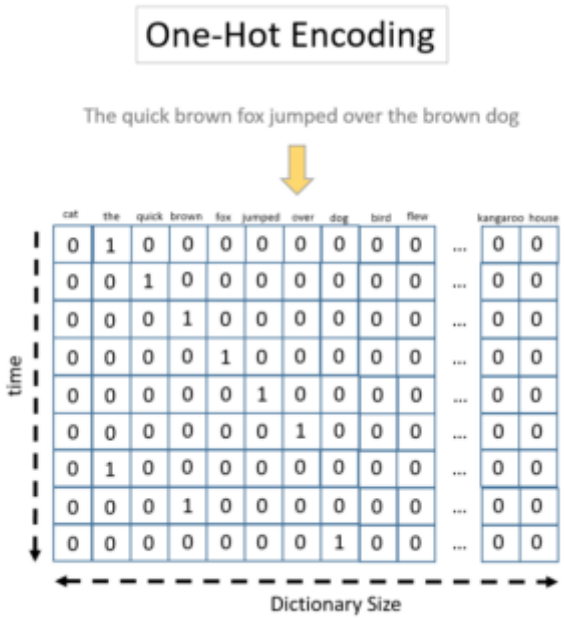

Figure 1.2: One hot representation of tokens

— **Third**, All tokens, now represented as dummy vectors $(xi)^n$ 1 where n denotes the number of tokens in the document, go through some big neural network. In this Projet Informatique, we first focus on pretrained **BERT Base Cased** , which has 12-layer, 768-hidden, 12-heads , 110M parameters 12 layers, 12 attention heads, 110M parameters and 768-hidden layers which is the dimensionality of the pooled layer. This network **bidirectionnally** "captures" the meaning of the sentence (from left to right and right to left) and embeds each of its tokens in a 768-dimensional vector space. Think of this as a transformation $f : \{0, 1\}^s \rightarrow R^{768}$ where $f(x)$ is the BERT representation of the token x

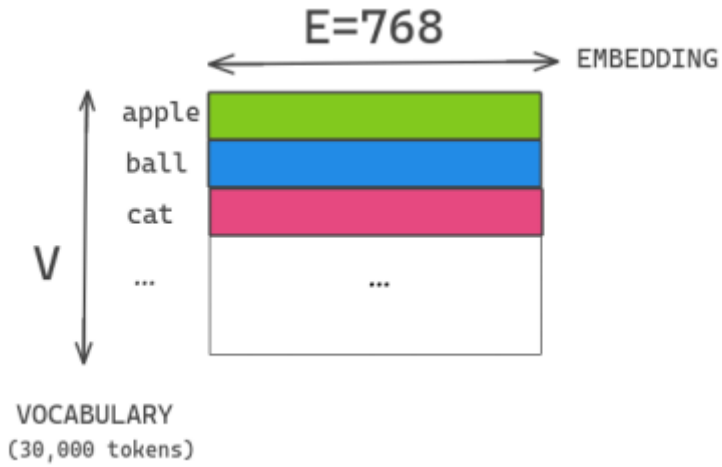

Figure 1.3: BERT's output

We have chosen **BERT base** rather than **BERT Large** which has 1024 hidden layers leading to an output leaving in R^{1024}, in order save time in the training phase especially when we notice that the dimensionality of BERT base output is 0.75 smaller than the dimensionality of BERT Large output. So, with our datasets containing 100000 samples, BERT base will not be as time-consuming as BERT-Large.

Fourth, for biclassification (Personal data vs REST), 'PER' label will be assigned to 1 and other labels to 0. For multiclassification, Each label will be a one-hot encoded vector living in R^{labels} where *labels*" designs the total number of labels present in the dataset. All the programming details are the file **anonymization.ipynb**

2 Bicalssification (Label 'PER' vs rest)

Given a token input $x \in R^{768}$ the objective is to determine whether It is a personal data or not. In our datasets, we have five labels ['O', 'PER', 'MISC', 'LOC', 'ORG'].
In this section 'PER' label will be assigned to 1 and other labels to 0.

2.1 METHOD1 : NEURAL NETWORKS (SUBPROBLEM 1)

2.1.1 A basic model

We used as a first model a two-hidden layer neural network without any non-linear activation between the different hidden layers. The output layer is activated with a sigmoid function :

$$f(x) = \frac{1}{e^{-x} + 1}$$

.

Given a token mapped in the 768-dimensional space, this classifier predicts the probability that this token reprensents a personal information. The bigger the output probability, the more likely the input is labeled 'PER'. Thus, we conclude that the threshold for decision is a **hyperparameter** to determine. But, in this project we settle for a threshold of 0.5.
Otherwise, we could **plot the precision-recall** curve for different thresholds and for a given precision, we choose the threshold that maximizes the recall.

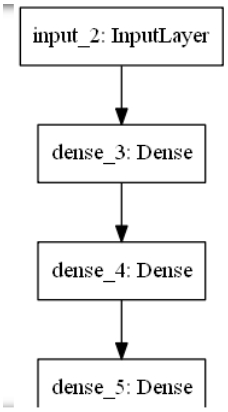

Figure 2.1: The NN plot

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 768)]             0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 4614
_________________________________________________________________
dense_4 (Dense)              (None, 12)                84
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 13
=================================================================
Total params: 4,711
Trainable params: 4,711
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.2: The NN summary

The loss function used is **the binary cross entropy** which stands for :

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Figure 2.3: Binary crossentropy

where $p(y_i)$ is the output predicted by the model, y_i the real label and N the total number of the training samples.

We chose this loss because It naturally pops up when trying to maximize **the likelihood** of the samples (vraisemblance in French) :
Let $L(y_i, x_i, W)$ the likelihood of our inputs x_i, y_i the real labels and W the weights used in the neural network. Maximizing $L((y_i)_{i\in\{1...n\}}, (x_i)_{i\in\{1...n\}}, W)$ is equivalent to maximizing

$$\sum_{i=1}^{n} logP(Y = y_i|X = x_i; W)$$

.
With a simple variable change $Z := 1_{Y=1}$ we obtain that :

$$\forall i,\ log\ P(Z = z_i|X = x_i; W) = z_i\ log\ P(Z = 1|X = x_i; W) + (1 - z_i)\ log\ P(Z = 0|X = x_i; W)$$

which is equal to the binary cross entropy.

As an optimizer, we used **Adam** to leverage the Stochastic gradient descent method and the momentum trick using not only the gradient at the iteration t but also previous gradients to adapt the learning rate at each iteration.

As metrics we use **precision, recall, accuracy, F1 score, F2 score and F5 score** which is more interseting than F1 score because we want to give more importance to recall than precision. Our objective is to detect all the personal data in the dataset and classify it correctly. So we should penalize low recalls with this F2 function.

$$F2 = \frac{5 \times precision \times recall}{4 \times precision + recall}$$

We trained the model for 20 epochs using a 8-batch size on **scaled** preprocessed data of CONLL2003 dataset.

We obtained good results :

```
precision is  0.9554139971733093
recall is 0.9230769276618958
accuracy is 0.9922000169754028
loss is 0.05284392833709717
area under ROC is 0.9803906083106995
f1_score is  0.9389671361502347
f2 score is  0.9293680297397771
f5_score is  0.9242801279772485
```

Figure 2.4: Results

2.1.2 Cross validation to tune the hyperparameters

We used grid searching (which uses K-Fold cross validation) to choose the best combination of hyperparameters among this list : {'batch size' : [12, 24], 'epochs' : [20,22], 'nbr units' :[6,8]}.

In grid searching, we chose the parameters giving a classifier with the highest F2 and F1 scores

The best estimator corresponds to $'batch_size' : 24, 'epochs' : 20, 'nbr_units' : 8 and leads to the following results :$

```
precision is  0.9585987261146497
recall is 0.9261538461538461
accuracy is 0.9926
f1_score is  0.9420970266040688
f2_score is  0.9324659231722429
```

Figure 2.5: Results after tuning the Hyperparameters

We see that the relevant scores have increased.

2.1.3 Training the model with weight balancing trick

Our dataset is highly imbalanced with favor to 'O' class. We have 4725 samples labeled 0 (not personal) and only 275 labeled 1 (personal). So if we balance it in a way, we may have better results.

There are three tricks that may be used :

— **Over-sampling** : Add more samples from under-represented class (PER).

— **undersampling** : remove samples from over-represented class.

— **weight-training** : penalize errors on minority classes.

We opted for the third trick because It does not require to change the input dataset unlike the other methods

The trick is to add some bias for minority classes, with adding weights to the two classes : we penalize the misclassification on the minority class. While calculating the loss we multiply each prediction by the weight of the class which is equal to

$$\frac{n_{samples}}{n_{samples_{inclass}} \times n_{classes}}$$

With this previous trick, we obtained good results but not better than grid search.

```
precision is  0.9318885448916409
recall is 0.9261538461538461
accuracy is 0.9908
f1_score is  0.9290123456790124
f2_score is  0.927295132470733
```

Figure 2.6: Results awith weight classes trick

2.2 METHOD2 : KNN CLASSIFICATION WITH C++

2.2.1 setting up

We decided here to use the Knn classifier that we implemented in TD 6. To do this, we first had to find a way to compile the c ++ codes on colab. Fortunately, colab already has a C ++ compiler by default. The challenge now was to install the ANN library on colab because the files implemented in TD use this library. To achieve this, we had to manually copy the compiled files from ANN to the different correspondents in the colab virtual machine.

Once we were able to run the TD6 files, the next step was to make the data compatible with our C ++ programs. To do this, you first had to change the files and labels. The label "PER" was replaced by 1 and all the others by 0. Then we joined the two test files in training into a single file whose first column represents the label.

Once the data file is ready, all you have to do is input the KNN program.

2.2.2 reminder of the KNN algorithm

Prediction with KNN is done in several steps :

— We use ANNsearch () to find the k nearest neighbor of the point we want to classify
— We calculate the mean of the labels of the k neighbors in question
— If the average is greater than 0.5, class 1 is predicted otherwise class 0 is predicted.

2.2.3 Results

We started by remarking that k was a local maximum because the F-score obtained with k = 3 was greater than those of its neighbors.
Thus with k = 3 we obtained an F-score of 0. 85 on testa and 0.83 on testb which was therefore lower than that obtained with the previous method.

```
execution time: 26660ms

                Predicted
                0       1
Actual  0       4654    21
        1       67      258

Error rate              0.0176
False alarm rate        0.00449198
Detection rate          0.793846
F-score                 0.854305
Precision               0.924731
```

Figure 2.7: result of testa

```
execution time: 26660ms

                Predicted
                0       1
Actual  0       4654    21
        1       67      258

Error rate              0.0176
False alarm rate        0.00449198
Detection rate          0.793846
F-score                 0.854305
Precision               0.924731
```

Figure 2.8: result of testb

3 Multiclassification (subproblem 3)

In our datasets, we have five labels ['O', 'PER', 'MISC', 'LOC', 'ORG']. The labels y_i will be represented as follows : $(1_{y_i='O'}, 1_{y_i='PER'}, 1_{y_i='MISC'}, 1_{y_i='LOC'}, 1_{y_i='ORG'})$ All the programming details are in the file **multi-classification-script.ipynb** which transforms a file of labels ('O', 'PER') into a one hot enocded file.

3.1 METHOD1 : NEURAL NETWORKS

We used the following neural network trained on scaled CONLL2003 dataset :

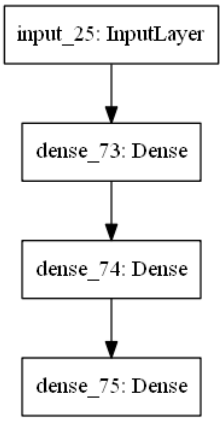

FIGURE 3.1: The NN plot

```
Model: "model_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_25 (InputLayer)        [(None, 768)]             0
_________________________________________________________________
dense_73 (Dense)             (None, 4)                 3076
_________________________________________________________________
dense_74 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_75 (Dense)             (None, 5)                 45
=================================================================
Total params: 3,161
Trainable params: 3,161
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 3.2: The NN summary

The output layer is five-neurone layer with a softmax activation :

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

FIGURE 3.3: Softmax

where s_j is the output probability that the input x is labeled j.

We obtained the following results which are a bit greater than the results found in the biclassification problem.

```
precision is  0.9523618
recall is 0.9476
accuracy is 0.95
loss is 0.24504836603030564
area under ROC is 0.9894135
f1_score is  0.949974947398449
f2_score is  0.948548556801311
f5_score is  0.9477822731070875
```

Figure 3.4: results for multiclassification

3.2 METHOD2 : KNN CLASSIFICATION WITH C++

3.2.1 setting up

Here we have modified the c ++ files used for the biclass classifier to take into account several classes so the setup is almost similar to that of the biclass KNN.
For the implementation of c ++ and ANN on colab, the procedure was exactly the same as for the biclass classifier.
For data prepartion, we first changed the labels of the test and training data. The label “PER” of 1, “O” becomes 2, “MISC” becomes 3, “LOC” becomes 4 and “ORG” becomes 5.
Then, just like in the biclass classifier, we merge the label data and the training data into a single file.

3.2.2 Modification of the KNN algorithm

To be able to predict multiple classes, we have made the following changes to the previous algorithms :

- We always calculate the k nearest neighbor of the target point
- Then we use a table to count the number of appearances of each class
- We predict the most frequent class

We also adapted the confusion matrix and the precision, error and recall calculations so as to consider several classes.

3.2.3 Results

We started with k = 5 and we obtained an extremely satisfactory score, ie an F-score = 0.991 for testa and F-score = 0.98 for testb.
Being quite satisfied, we chose to decide to stay on k = 5. Thus with k = 3 we obtained an F-score of 0. 85 on testa and 0.83 on testb which was therefore lower than that obtained with the previous method.

execution time: 23382ms

		Predicted				
		1	2	3	4	5
Actual	1	289	18	5	6	7
Actual	2	8	4127	12	4	6
Actual	3	1	43	62	7	2
Actual	4	12	26	42	113	13
Actual	5	14	79	20	23	61

Error rate class 1	0.00829342
False alarm rate class 1	0.00795816
Detection rate class 1	0.991352
F-score class 1	0.991471
Precision class 1	0.991591

Figure 3.5: result of testa

execution time: 23382ms

		Predicted				
		1	2	3	4	5
Actual	1	289	18	5	6	7
Actual	2	8	4127	12	4	6
Actual	3	1	43	62	7	2
Actual	4	12	26	42	113	13
Actual	5	14	79	20	23	61

Error rate class 1	0.00829342
False alarm rate class 1	0.00795816
Detection rate class 1	0.991352
F-score class 1	0.991471
Precision class 1	0.991591

Figure 3.6: result of testb

4 Testing the NN model on other datasets (sub-problem 4)

In the files "evaluate_model_on_BTC.ipynb" and "evaluate_model_on_wikigold.ipynb".We tested our best neural network classifiers (the best biclassifier and the multiclassifier) on two other datasets : BTC and wikigold dataset.

We carried out the exact same steps of the "Steps before classification" section on these two datasets.

Here are the results :

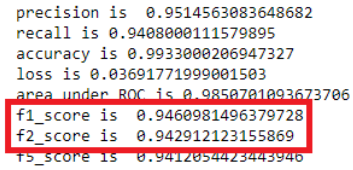

FIGURE 4.1: evaluating of the Biclassifier on BTC

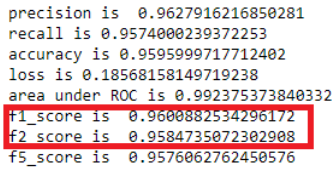

FIGURE 4.2: evaluating of the multiclassifier on BTC

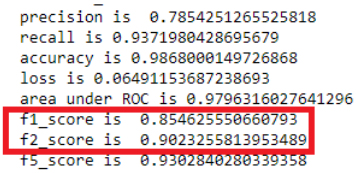

FIGURE 4.3: evaluating of the Biclassifier on Wikigold

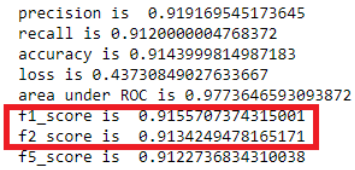

FIGURE 4.4: evaluating of the multiclassifier on Wikigold

For BTC, the results are similar to CONLL2003 dataset results. For Wikigold, the results are good but worse than CONLL2003 dataset results. As we notice, It is due to a low precision score (around 0.78) which means that there are a lot of false positive predictions. This may be explained by the fact that Wikigold is a dataset taken from Wikipedia, so its vocabulary size is far larger than CONLL2003 dataset and we easily can find two vocabulary words which are close in meaning (so close in the embedding representation) but one is Personal and the other is not. Example :the word 'PETER' will be tokenized in PET and ER. In this case, PET should take 'PER' label. whereas, the word 'PET' should take 'O' label.

A conclusion is that the more the vocabulary size increases, the harder for our biclassifier to make precise predictions.

5 Conclusion

From the above we conclude that the KNN multiclassifier had better scores than the neural network multiclassifier (with the chosen parameters which could be more tuned) in the multiclassification task whereas, in the biclassification task, the neural network biclassifier has beaten the KNN classifier.