PI9 - REPORT

INF442-9 - GDPR in practice: data anonymization

May 19, 2020

ALBUQUERQUE SILVA Igor, GALVÃO LOPES Aloysio
X2018

Contents

1 Abstract

This project consists in the implementation of several classifiers to solve the problem of text data anonymization. In the first moment, It was given a BERT [1] pretrained model that converted the texts into a numerical vector, and we developed 9 different classifiers to compare results using 5 different datasets. In a second moment an algorithm was developed to read the text data and do the classification alone. It was found that a neural network approach achieved the best results.

2 Tasks

The group has chosen to do the four sub-problems proposed in this exercise. The code produced is available in a GitHub repository (`https://github.com/aloysiogl/pi_inf442`), where instructions for its use can also be found. For each task we did the following:

- **Task 1:** We implemented a KNN classifier, a Linear Regression for classification, a Logistic Regression for classification and a Neural Network for classification. Those three classify the tokens from BERT's [1] output.
- **Task 2:** We implemented our own encoder which utilises regex for tokenization and our classification is made via Neural Network.
- **Task 3:** We implemented an OVR classifier and extended the KNN functionality, as well as a multi-label neural network output.
- **Task 4:** We utilized the provided python code with some adaptations to generate new BERT tokens for 3 new datasets.

3 Predictors

In this section we will briefly introduce the different predictors we used. The details seen in the course will not be emphasized and the main focus will be to describe the particularities of the implemented predictors, as well as analyze the effectiveness of such predictors.

We use the convention d = dimension of the data set, n = numbers of samples and κ = number of classes.

3.1 KNN Classifier

The idea behind a KNN Classifier is really simple: to label a new feature, it must take the data from its K-nearest neighbors and derive the output based on them. A distance is considered as the euclidian distance, and the method used to deduce the label is to get the one which has maximum sum of the inverse of distances.

To calculate efficiently the K-nearest neighbors, the ANN [2] library in C++ was used, which has an approximate kd-tree implementation. An attempt was made to plot an ROC curve in order to determine the best value of K for the problem, but due to the complexity and unbalance of the dataset, no conclusions were made since the graph didn't follow any visible pattern. We arbitrarily chose K = 5 as default.

3.2 Linear Regression

This predictor is a C++ implementation of the Linear Regression classifier seen in course. For the training part it utilizes the equation : $B = (X^t X)^{-1} X^t Z$.

X and Z use the same definitions seen in the course. Analysing the operations in the expression above and considering the naive algorithm for matrix multiplication and Gauss-Jordan for matrix in version we conclude that the complexity of the training step is $\mathcal{O}(d^2 n + d^3 + d^2 \kappa)$.

Each classification of a sample v is computed as $argmax([1v^t]B)$, this computation can be done in $\mathcal{O}(d\kappa)$. This algorithm is already adapted for binary and multi-class classification.

3.3 Logistic Regression

This predictor is a C++ implementation of the Logistic Regression for binary classification seen in course. The logistic regression assumes that the data is distributed using a logistic distribution and has the propriety of estimating the real probability of each class, in this way it generally outperforms the Linear Regression. In this case $\kappa = 2$ and is a constant.

The training process consists of maximizing a likelihood function ℓ of an estimator vector β analogous to the B matrix in the linear regression. More details are explained in the course. For this we use Newton-Raphson's method which needs the evaluations of $\nabla\ell(\beta)$ and $(\nabla^2\ell(\beta))^{-1}$. The evaluation of $\nabla\ell(\beta)$ is $\mathcal{O}(dn)$ and the evaluation of $\nabla^2\ell(\beta)$ is $\mathcal{O}(d^2n + d^3)$. Therefore the complexity for each step is the same as the Linear Regression (with $\kappa = 2$). We limited the number of iterations to 10 due to time constraint limitations.

The classification step for a vector v consists on $\sigma([1v^t]\beta)$ which is $\mathcal{O}(d)$.

3.4 OVR Multi Class Classifier

The OVR (one vs rest) classifier is a C++ implementation that generalizes binary classifiers to multi class classifiers. The only restriction is that the classifiers utilize some non discrete metric for classification.

This classifier works in the following way: for each class c in a data set D it generates a binary data set D' where all the samples are the same and the labels are: 0 if c and 1 else. Then we take a classier (in this case only the logistic or the linear regression) and run the training for each D'. Therefore training takes $\mathcal{O}(\kappa T^t)$ where T^t is the training time in the selected method.

For the classification we take a similar approach and assign the class with higher score in binary classification, therefore we ran the classification κ times. This takes $\mathcal{O}(\kappa T^c)$ where T^c is the classification time in the selected method.

3.5 BERT-NN Classifier

This classier is a Neural Network implemented in python using PyTorch [3]. We utilized three linear layers, the ReLU activation function before each inner layer and the sigmoid activation function for the last layer. The considered loss function was the cross entropy loss with a relative weight 10 to the prediction of a person name entity. Is important to note that this network takes as input the BERT processed outputs.

The training process complexity is the complexity of back propagation and the classification is the complexity of three matrix multiplications.

An attempt was made to try to integrate this Network in BERT in a single Network but the training time and possibly the wrong choice of the cost function prevented the attempt from yielding better results than the separate approach.

3.6 REGEX-NN Classifier

This classifier uses the same approach as the BERT-NN but this time we generate some tokens based on regex. Firstly we take the input and transform it in a single line text separated by spaces. Then we take a list of r different regex templates and for each word we check if the regex applies. After this process we represent each word as a binary vector r dimensions such that the i-th entry represents that the word satisfies the i-th regex.

We then use a Neural Network similar to the one in the BERT-NN classifier and generate the classification for each work. The time to generate the tokens is the sum of the times for the regex string processing form all the given tokens. As this varies largely on the kind of template used ad the templates are modifiable it's not possible to provide a reasonable worst case complexity.

4 Results

4.1 Metrics

We chose three metrics to evaluate our classifiers: sensitivity (recall, true positive rate or detection rate), f-score, and the sum of training/building time and classification time of the complete test dataset. For the multi-label classifiers, the confusion matrices were binarized to calculate the sensitivity and f-score values.

We chose sensitivity because a high-sensitivity value in this problem indicates that there are few personal words that will not be anonymized, which is more critical than non-personal data being anonymized.

4.2 Datasets

Five datasets were used to evaluate our classifiers.

- English Dataset A/B: CoNLL2003 [4] english dataset with each test.
- Spanish Dataset: CoNLL2002 [5] spanish dataset with test A.
- Dutch Dataset: CoNLL2002 [5] dutch dataset with test A.
- Portuguese Dataset: LeNER-Br [6] dataset, which consists of brazilian legal text.

We did a comparison of all the methods only in the English datasets. For the three other languages, we compare only three models: the best of our "classic" classifiers, which is the OVR Logistic, the best classifier, which is the Bert-NN and our custom-made Regex-NN classifier.

4.3 Data

We used two types of machines to calculate our results: for the Bert-NN classifiers a Kaggle machine was used, which has a Tesla P100 GPU. For all the other methods, we used four different AWS c5.xlarge machines, which have up to 3.6 GHz clock speeds.

Figures 1 to 5 contain the results for each dataset. Note that the values start at 0.2 for better visualization. The time scale used is the natural logarithm of the normalized time in seconds. The classifiers are labeled in two types: BIN and NER, which represent if we used binary or 5-class labels.

Figure 1: Prediction results in the English Dataset A, separated by metric.

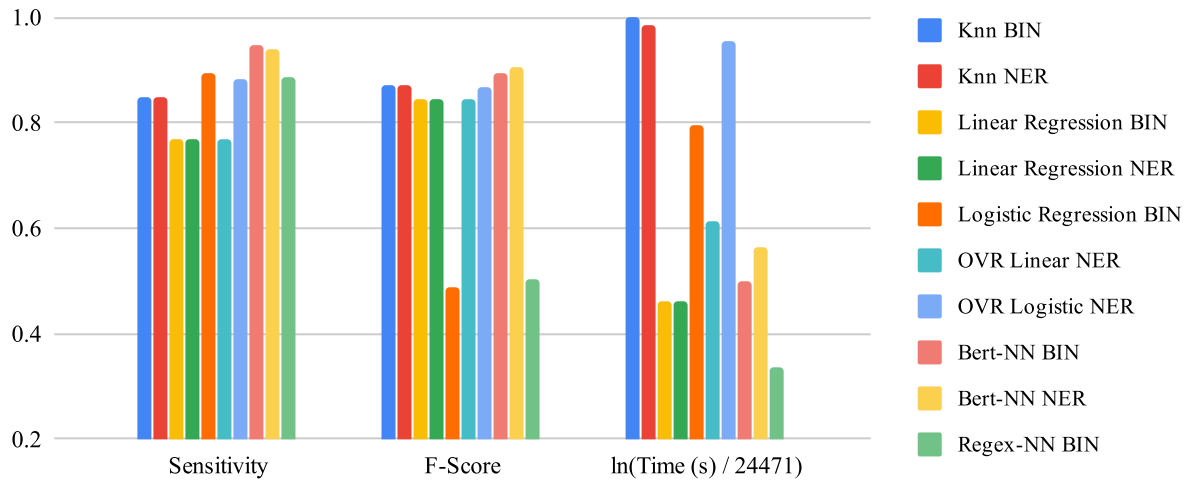
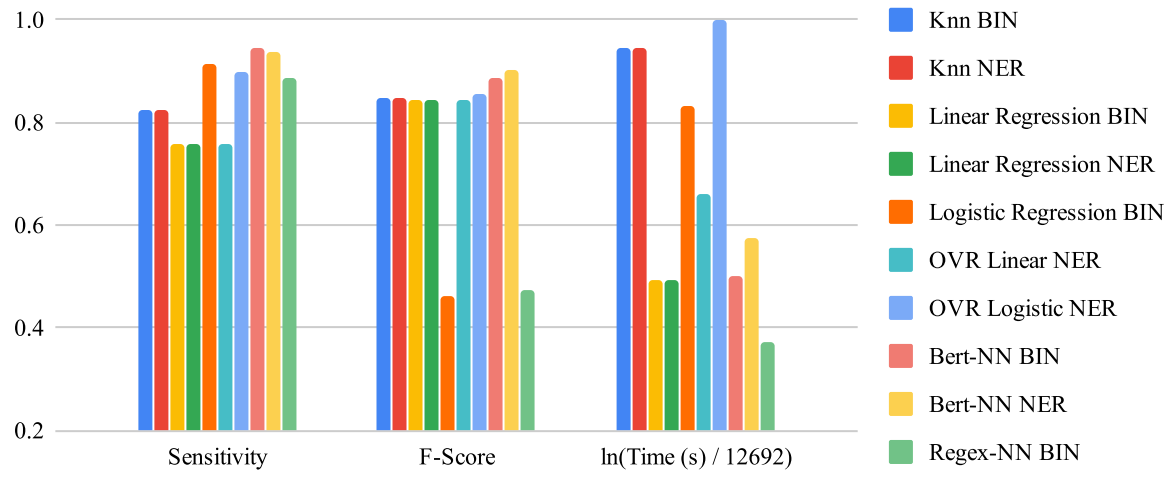


Figure 2: Prediction results in the English Dataset B, separated by metric.



Several conclusions can be taken from the English datasets. The best classifier is the Bert-NN, which also has one of the lowest processing times. The Linear Regression showed the poorest results, but the second-best processing times. The Logistic Regression with binary classes presented good sensitivity, but the worst f-score of all, and a really high processing time. This f-score was improved with multi-classes. The Knn had the worst processing time and an average result. Our Regex-NN method is incredibly fast, and its performance is comparable to the Logistic Regression with the BERT preprocessing.

We can also conclude that all the multi-class methods are inferior or equivalent in terms of sensitivity, but are superior or equivalent in terms of f-score. We can explain this fact by since they are biased for sensitivity instead of precision in binary classes, the multi-class model is biased for sensitivity in every class, balancing the confusion matrix and increasing precision.

Figure 3: Prediction results in the Spanish Dataset, separated by metric.

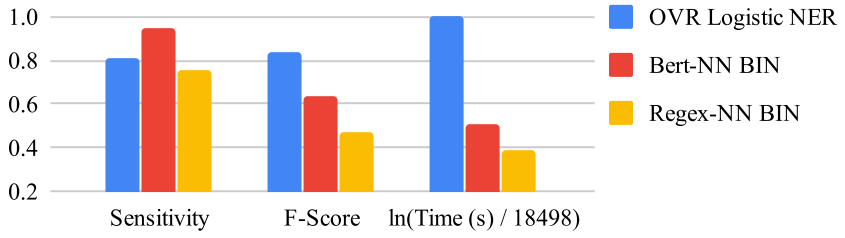


Figure 4: Prediction results in the Dutch Dataset, separated by metric.

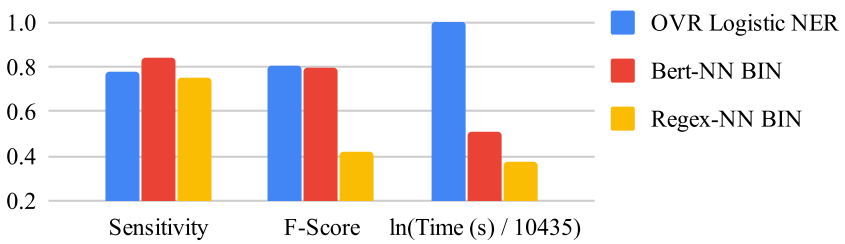
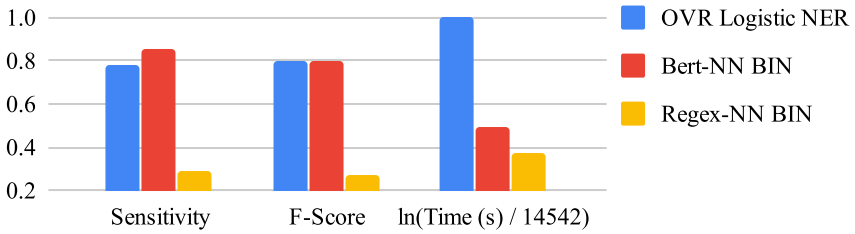


Figure 5: Prediction results in the Portuguese Dataset, separated by metric.



Results were worse on the datasets from other languages. This can be explained by two reasons. First because the loaded BERT [1] model was trained using English text. Second because these datasets are considerably more unbalanced than the English datasets (6.1% of PER labels in English A, 6.0% in English B, 3.9% in Spanish, 3.0% in Dutch and 1.5% in Portuguese). The Regex-NN method was especially affected by this fact, which is an evidence that it is probably overfitting. We also have to consider that those Regex rules might not work as well in other languages as in English.

5 Conclusion

We were able to successfully evaluate 10 different methods of classification. We even obtained better results than those from the CoNLL2003 [4] competition champions with some of them.

An interesting finding was that all the multi-class methods were inferior or equivalent in terms of sensitivity, but are superior or equivalent in terms of f-score, which was explained in Section 4.3.

Our Regex-NN method is incredibly fast, and its performance is comparable to the Logistic Regression with the BERT preprocessing in the English dataset, but its performance decreases considerably in other languages, which can be explained because these datasets are considerably more unbalanced than the English datasets.

Future work in this problem can include an analysis of ways to adapt for the unbalance of datasets, further iterations on the neural network models and increase the number of Regex expressions for the custom classifier.

References

[1] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805 [cs]* (May 24, 2019). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 05/18/2020).

[2] David M. Mount and Sunil Arya. *ANN: A Library for Approximate Nearest Neighbor Searching*. Jan. 27, 2010. URL: http://www.cs.umd.edu/~mount/ANN/.

[3] *PyTorch*. URL: https://pytorch.org/ (visited on 05/19/2020).

[4] 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. Edmonton, Canada: Association for Computational Linguistics, 2003, pp. 142–147. DOI: 10.3115/1119176.1119195. URL: https://doi.org/10.3115/1119176.1119195.

[5] Erik F. Tjong Kim Sang. "Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition". In: *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*. COLING-02. USA: Association for Computational Linguistics, 2002, pp. 1–4. DOI: 10.3115/1118853.1118877. URL: https://doi.org/10.3115/1118853.1118877.

[6] Pedro Henrique Luz de Araujo et al. "LeNER-Br: A Dataset for Named Entity Recognition in Brazilian Legal Text". In: *Computational Processing of the Portuguese Language*. Ed. by Aline Villavicencio et al. Vol. 11122. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 313–323. ISBN: 978-3-319-99721-6 978-3-319-99722-3. DOI: 10.1007/978-3-319-99722-3_32. URL: http://link.springer.com/10.1007/978-3-319-99722-3_32 (visited on 05/19/2020).