REPORT DEEP LEARNING

Pietro Volpato and Francesco Chemello

Introduction

This project is about the classification of viral and not viral DNA strings using a deep neural network and it takes inspiration from the **ViraMiner** project, where the whole code is available on **GitHub**[1].
The general idea we adopted was to combine two different types of neural networks in an ensemble learning environment to improve overall performance due to the ability of the two networks to extract different information from the same dataset. We implemented a *CNN*, two different *RNN* networks (a *LSTM* and a *GRU*), and some meta-classifiers such as *SVM, Random Forest and Gradient Boost Classifier*.
For running the project we used **Google Colab**[3] since the computational power required for running all the code in reasonable time is high. We uploaded the full dataset in Google Drive and used it for running the project.

Dataset

Upsample and SMOTE

The dataset used is the same as the original **ViraMiner** project which is available on **GitHub**[2] and it is already divided into training, validation and test.
We noticed that in the training dataset there was a great **imbalance** between the two classes: 206772 samples of class 0 and only 4466 of class 1. In order to overcome this problem, first we added a **class weight**[13] in the loss function to guide the training of the two networks. Then, we implemented a basic oversampling using the sklearn's **function resample**[12] to upsample the minority class 1 to reach 40000 samples. Finally, we decided to also implement a more accurate oversampling function thanks to the **SMOTE**[11] **algorithm,** provided by the imbalance-learn library. Doing that, we obtained 80000 samples of class 1 (that is around 27.5% of the whole training dataset).
The impact on the overall performance will be done in the **Results section**.

One-Hot Encoding

After the oversample of the minority class, we convert the 3 dataset (training, validation and test) into the one-hot encoding, to be able to use alphabetical data in the convolutional neural network. We implemented two functions called ***onehot_encoder*** and ***dna_onehot_encoder*** that, starting from a string of DNA, returns its one-hot representation (we take inspiration from the ones implemented by **ViraMiner**).
Those functions return a tridimensional array of shape (n_samples, sequence_lenght, 4), where **n_samples** is the total number of samples in the dataset that is different for the 3 cases, **sequence_lenght** is the length of a sequence of DNA that is 300 in our case and **4** is the number of nucleotides (**A**denine, **C**ytosine, **G**uanine and **T**hymine).

Models

We considered 3 kinds of networks: **CNN**, **LSTM** and **GRU**. We choose to adopt the RNN because of their ability to learn patterns, i.e. to recognize viral DNA by finding some shared features in all the viral ones. LSTM and GRU are the most popular RNN and they are able to achieve good performance thanks to their internal structure. We also implement the CNN because it is able to perform well in different scenarios.
Then, we implemented a **meta-classificator** to perform the actual classification of the data.

Structure

Let's analyze the overall structure:

- The **CNN** net is organized as follows: five layers with one convolution, one batch normalization, one PRelu activation function, one pooling and one dropout each.
- The **LSTM** net has three convolutions and one pooling for reducing the computational time and then one LSTM layer with 128 hidden features and 6 layers. The LSTM is

bidirectional in order to boost its capability to learn patterns.
- The **GRU** structure is similar to the LSTM, it has one convolution and one pooling to reduce the computational time and then one bidirectional GRU layer.

All the three networks end with two fully connected layers.

Ensemble and Meta-Classificator

For doing the final prediction, we combined 4 meta classifiers *(formal definition is provided)*:

- **Support Vector Machine (SVM)**[6]: A supervised learning algorithm that finds the optimal hyperplane which best separates the data into classes.
- **XGBoost (Extreme Gradient Boosting)**[7]: An optimized gradient boosting algorithm that builds an ensemble of trees in a sequential manner. It uses advanced regularization techniques and efficient computation to enhance performance and speed.
- **Random Forest**[8]: An ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. It improves accuracy and controls overfitting.
- **Gradient Boosting Classifier**[9]: A machine learning technique for regression and classification problems that builds an ensemble of decision trees. Each new tree corrects errors made by the previously trained trees, and they are combined to improve overall accuracy.

The final classification is done using the **stack ensemble learning**[4] so, for each of those meta classifiers, we give as input for the training the prediction of the CNN and one of the RNN and then we obtain the final classification.
The results are reported on the tables in the **Results section**.

Result & Discussion

After the test phase, we have calculated the accuracy, precision, recall, F1-score and AUROC using the **sklearn library**[14 15 16 17 18].

The result* can be seen in the .xmls file attached in the submission.
We deal with several issues regarding the dataset: it was imbalanced and contains much more samples of class 0 than class 1. We tried to fix that by implementing the upsample and SMOTE and we compare the different results with different architectures. We obtain good results in terms of accuracy, precision and recall but we always obtain a discrete AUROC. This could be explained as the fact that the networks are not able to discriminate the classes and it could be a problem of the dataset we used that is imbalance and also with the technique used the performance does not improve.
In conclusion, we noticed that the ensemble architecture did not boost the performance so much but it helps to keep the performance high, especially when one of the sub-net (CNN and the RNN) did not perform well. Also, we can see that SVM performs particularly well in almost every scenario, different from the others where they perform well only in particular configurations.

*note that results can be different due to the ***shuffle*** in the train loader.

References

1. https://github.com/NeuroCSUT/ViraMiner
2. https://github.com/NeuroCSUT/ViraMiner/tree/master/data/DNA_data
3. https://colab.research.google.com/
4. https://h2o.ai/wiki/stack-ensemble/
5. https://scikit-learn.org/stable/modules/svm.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
7. https://xgboost.readthedocs.io/en/stable/python/python_intro.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
9. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
10. https://pytorch.org/docs/stable/index.html
11. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
12. https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html
13. https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html
14. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
15. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
16. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
17. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
18. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.htm