HPC-AI Mines Paristech

Machine Learning Project

MILLAN Aymeric, VITAL Eduardo, Team AE Dragons

Professor: Nelle Varoquaux

February 27, 2022

1 Introduction

The Machine Learning Project consists of a prediction challenge, hosted by the CodaLab platform, for inferring protein functions; more specifically, whether a protein of a particular organism has a particular function. During the project, various machine learning techniques were tested, from data processing to training algorithms, using the *Python* programming language and favouring the *scikit-learn* library.

The goal of this data challenge itself is to resolve a use case in data analysis which is inspired from a real scientific project (and real data in a simplified form), where the features were extracted from protein sequences of about 300 organisms. Data observation and solution were made available through *protein_train* files, in which the model should be trained. After that, the model could predict by means of a *protein_test* and *protein_valid* data sets and be evaluated internally (facilitated by a submission script), in order to output a final score. This final score is sent to a ranking, defining, thus, the competition challenge.

Our code can be found in this repository GitHub, you will find more informations on how we processed to get our final models in the next sections.

2 Data Analysis and Visualisation

The first step of the project is to comprehend the data. The training set contains 53118 observations and 952 features, while the output is binary. We can use PCA to reduce the dimensionality of the data in order to do a 3d scatter plot of its 3 most important principal components coloured by its label [figure 1a]. From this graphic, we can promptly see that each label is not linearly separable from each other, inferring that linear models, such as logistic regression or SVM with a linear kernel, will not do a good classification job. The same conclusion can be achieved through the figure 1b showing the 5 principal components correlations and the scaling effect on the distribution.

3 Pipeline

After understanding our data, we can begin to work on it. As our training base is the only one we have with a corresponding solution, in this first moment it is the only way to also test the performance of the algorithms. However, by submitting a model, we will also receive a score from

the test and validation set using the *balanced accuracy score* metric (metric used during all the project).

If we didn't have those extra datasets, we would have to further devise our training observations into another subset in order to test the performance of the algorithm. We chose, nevertheless, not to do this with the intent to optimize our use of data; computing, thus, the score only in the submitting stage.

In order to properly choose a model, a series of operations have to be applied, including:

- Pre-process the data.
- Grid-search different estimators to optimize its hyper-parameters.
- Submit the best model from each Grid-search.
- Analyse the best score obtained from the different models, already ranked in the platform.

The first two steps can be joined together in a *scikit-learn* pipeline as a means to fit and predict the series of algorithms in only one step.

3.1 Pre-processing

As observed, the data has a higher number of features, so it would be a good idea to implement a dimensionality reduction to have lesser features and avoid outfitting. Moreover, scaling techniques are of the most importance to assure that features have the same length.

Therefore, due to the above points and profiting from the fact that some dimension reduction methods require scaling first, we can build a small pre-processing pipeline consisting of scaling and feature reduction.

There are, however, plenty of possibilities for these pipelines using different algorithms for each step. In addition, each one of those pre-processing algorithms can have multiple parameters (hyper-parameters). Our approach to choose, thus, the final pre-processing pipeline, was to grid-search through all these pipelines, but at the end, instead of optimising the hyper-parameters of various predicting estimators, we used only one simple model.

This approach is much less costly and can achieve good results in the choice of the pre-processor. The simple model chosen was a KNN with specific parameters that presented a good performance in previous tests. Other options discussed were decision tree and logistic regression.

For the pre-processing algorithms, the options tested were: Standard Scaler, Robust Scaler, Quartile Transformer, Kernel PCA, Feature Agglomeration and Gaussian Random Projection, resulting in 9 pipelines. Those operations were run locally due to an issue in the CEMEF cluster we are presenting in the next section.

The best results obtained, used in the main pipeline was a robust scaler with feature agglomeration (2 illustrates the performances), clustering 650 features.

3.2 Optimisation of Hyper-parameters

To optimize the hyper-parameters of the different models, we used a grid-search, or even random grid-search, approach. This method was applied for the pre-processing pipeline and also to the training of each estimator. This process was used specifying k-fold cross-validation training, using k=4 usually.

One important point during the utilisation of the grid-search function is to use the pipeline object: fitting the scaling, the dimensionality reduction and the machine learning algorithm at

each train set of the cross-validation and applying it to the corresponding test subset. Otherwise, without the use of a pipeline, the fitting of the pre-processing would be done in the whole data set, causing over-fitting in the test subsets of the cross-validations contained in the grid-search.

Nevertheless, this faulted approach was used to enable the use of the CEMEF cluster, where more expensive calculations could be run. This had to happen because, due to memory and C++ libraries issues, a grid-search using a pipeline crashed when run on the cluster.

This grid-searches were trained with different algorithms: Boosted trees (from xgboost), random forest, bagging trees, decision trees, KNN, SVC, neural networks and logistic regression.

Each one of the best models is then trained in the new validation and test data sets for submission. One important point, is that the pre-processing applied in these observations was fitted only in the training ones.

The best of the models submitted can be viewed in CodaLab ranking. Over the best models found, another fine tuning was yet applied in the random grid-search realized in the training set, with a Poisson distribution centered in the best parameters of the estimators. Then, those new models were submitted, achieving a higher score.

4 Results and Analysis

The best overall models were the Neural Network (Multi-Layers Perceptron Classifier) and the KNN classifiers. They were combined with the pre-processing pipeline using a robust scaler and feature agglomeration to reduce the dimension of the data. In this way, a final score of around 0.93 could be obtained.

The dimensionality reduction algorithm varied a lot concerning the number of features chosen, probably because of the limited search space constrained by the computational coast. The PCA with poly kernel, for example, took a very long time to fit and transform, so the search space had to be very limited.

Moreover, we can analyse our models with a leaning curve [figure 3a and 3a]. As we can see, for one of our MLP and KNN models, we have an expected convergence between our training and cross-validations scores. The gap in the first one suggests a small overfit, while in the second case this is less visible. Analysing the curve tendency, however, shows us that, for both models, performance could be improved if more data were to be acquired.

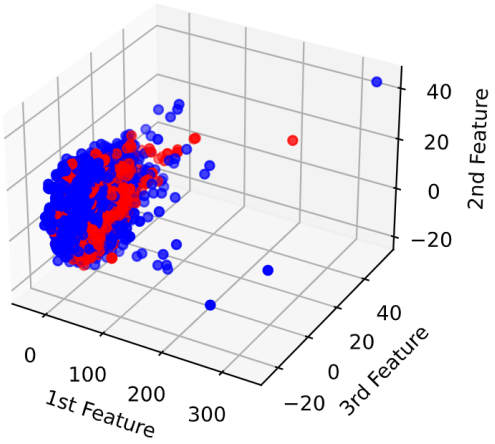
Our final models with the best parameters can be found in the *main.py* file. Files in the *models* folder are the ones we ran on the CEMEF's cluster. Finally, you can find in figure 4 the performance of other models we have tried. These are the scores before fine tuning the hyper parameters and were used to compare the effect of scaling and dimension reduction before fitting. The number after the name of the transformer used for the reduction corresponds to the main hyper-parameter used (e.g. *n_clusters* for FeatureAgglomeration). These scores are based on local cross validations, thus not involving the codalab score. When no score is displayed, it means that the estimator did not converge in a reasonable time.

5 Conclusion

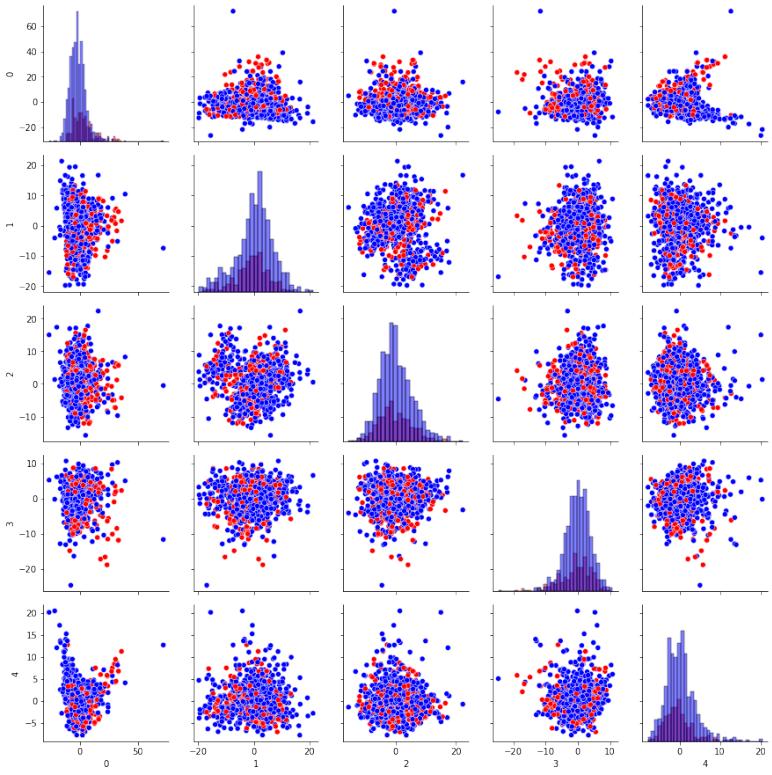
In the beginning of the project, our approach consisted in trying different models in order to find the best one. Our strategy evolved when we began to simplify some algorithms in order to better understand the data and quicker run tests on it, always prioritizing the use of the CEMEF cluster nevertheless.

In this way we were able to find a good pre-processing pipeline, resulting in better models than previously tested. Those estimators were improved by grid-searches and fine tuned after, always making the effort not to fit any algorithm in the test datasets, for the purpose of achieving the highest score.

A Figures



(a) 3 components PCA representation: blue=0, red=1.



(b) 5 components PCA compared 2x2

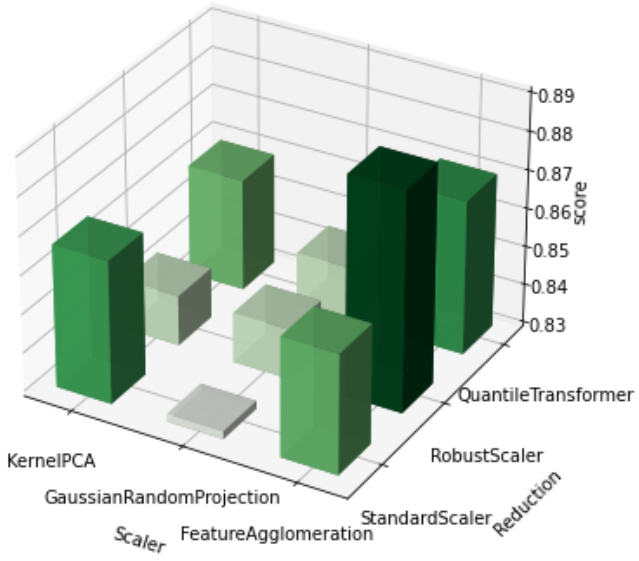
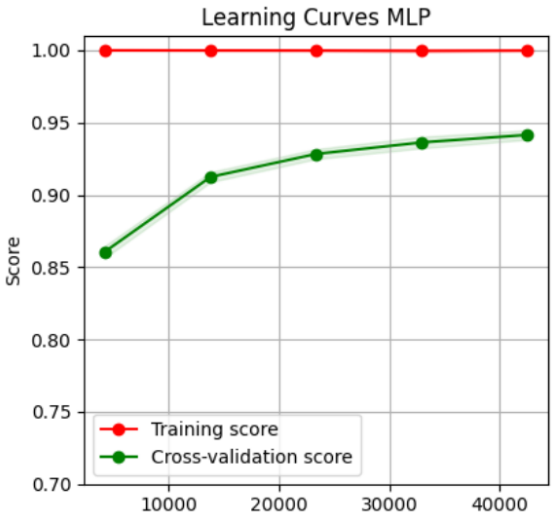
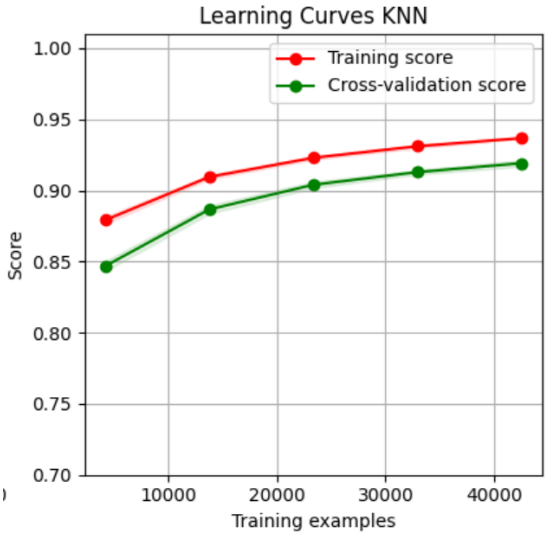


Figure 2: Pipeline Scores



(a) Learning curve MLP



(b) Learning curve KNN

Figure 4: Models Performance Table

Scaler	**Reductor**	**Model**			
		MLP	KNN	XGBOOST	SelfTrainMLP
QuantTrans	FeaturAgg (650)	0.893	0.892	0.871	0.88
	GaussProj (700)	0.892	0.858	X	0.891
RobustScaler	GaussProj (800)	0.896	0.873	X	0.900
	PolyPCA (423)	0.778	0.89	X	0.784
	LinPCA (177)	0.887	0.879	0.844	0.885
StandardScaler	FeaturAgg (100)	0.878	0.893	0.868	0.879
	GaussProj (700)	0.896	0.85	X	0.895
	PolyPCA (423)	0.772	0.898	0.873	0.773