University of Liège

Activity prediction for chemical compounds

Introduction to machine learning

Maxime Meurisse (20161278)
François Rozet (20161024)
Valentin Vermeylen (20162864)

MSc in Computer Science and Engineering
Academic year 2019-2020

1 Introduction

The project is organized as a competition between different teams on the Kaggle platform. As mentioned on this website, it consists of using (supervised) learning techniques to design a model able to predict the activity of a chemical compound (described by its molecular structure). The main goal is about *determining the ability for a chemical compound to inhibit HIV replication.* [1]

Since the prediction associated with each molecule is a boolean (the molecules inhibit HIV replication or not), we tried classification methods. We used the methods seen during the course, but also other methods found in scientific papers or websites. We also took inspiration in this scikit-learn flowchart to guide our tests.

The main classifiers we focus about were the *K-Neighbors Classifier*, *Multilayer Perceptron*, *Random Forest Classifier*, *Support Vector Machine* and combinations of those using *Ensemble* and handcrafted methods.

First of all, we tested each method individually without changing any parameters to get an idea of their performance (cf. section 3). Then we tried to optimize the parameters of each one by iteratively applying them on the learning set and estimating their accuracy. Since the results weren't satisfying enough both with for local estimation and for the kaggle platform, we had the idea of combining them which lead better results.

Apart from the actual classifier algorithm, we also tried several fingerprint representations of the SMILES molecules. For example, we tried the following : *Morgan*, *RDKFingerprint* (Daylight-like), *Avalon*, *MACCS* fingerprints and a few others. Each of them have parameters that we tried to tweak for the better. However, we only realized that there were different fingerprints near the end of our journey which did not left us with a lot of time to experiment with them.

2 Data pre-processing

In order to train our models, we received a training set consisting of 15 939 chemical compounds with their corresponding activity. Each chemical compound is described under the SMILES format (a line notation for the chemical structure of molecules). However, this format is difficult to use as it is by our models[1]. We therefore transformed the SMILES into binary fingerprints (cf. section 5), i.e. vector of zeros and ones describing features.

Most of the time these fingerprints are sparse, therefore it is possible that one of the feature is the *same* for all molecules. Therefore we applied a variance threshold [2] to remove all features whose variances are null.

3 Estimation of the AUC

To evaluated the accuracy of our models we used the ROC AUC metric (area under the ROC curve) available in the scikit-learn library [3] combined with a random permutation cross-validator [4].

[1]Afterwards, we think that we could have tried text analysis algorithms.

The latter works as follow : a random permutation of the learning set is computed. The model is trained on a $1-x$ portion of the permutation and used to predict the class of the remaining x portion. Then the AUC score is computed using the prediction probabilities and the actual classes. This is repeated n times. Finally, the n gathered scores are averaged.

In contrast to the classic *KFold* cross-validator, this one dissociates the number of splits (n) and the test size (x). In our case we chose $x = 0.25$ and $n = 5$.

The comparison between the estimated AUC and the actual (private) AUC will be conducted in section 6.

4 Models

As said in the introduction we first tested the k-neighbors classifier (KNN), multilayer perceptron (MLP), random forest classifier (RFC) and support vector machine (SVM) algorithms individually with the original Morgan fingerprint. We also tried decision tree classifier (DTC) and linear discriminant analysis (LDA) but they were a lot worse than the others.

The estimations gravitated around 74 %, but when we submitted these algorithms, the results were quite poor (~71 %) in comparison. We then realized that using the `useFeature = True` parameter significantly increased the score for both our estimations and Kaggle scores. Since we don't know much about organic chemistry, the consequences of this parameter are kind of blurry in our mind. Yet from what we understood, when set to `True`, the algorithm will detect feature-based invariants (predefined patterns) within the molecule [5].

At the same time, we noticed the parameters `nBits` (the number of bits in the bit vector) and `radius`. But, since we didn't want our scripts to run for too long, we didn't increase the number of bits until the end of our parameters testing phase, which was unfortunate as increasing that parameter usually provided better results (but induced computation times that were much longer).

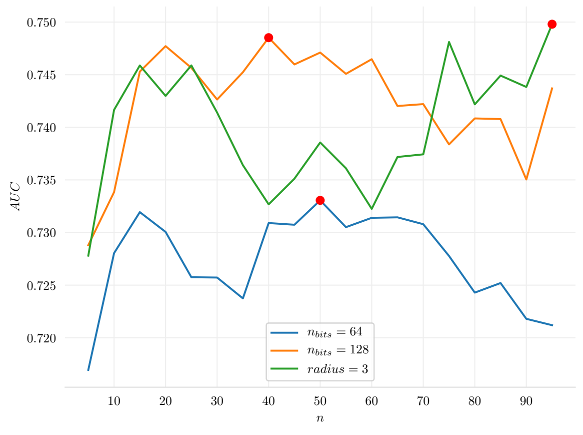
In the following, if it not precised, the fingerprint used is Morgan, the number of bits 128, the radius 2 and the parameter `useFeature` is `True`.

4.1 K-Neighbors classifier

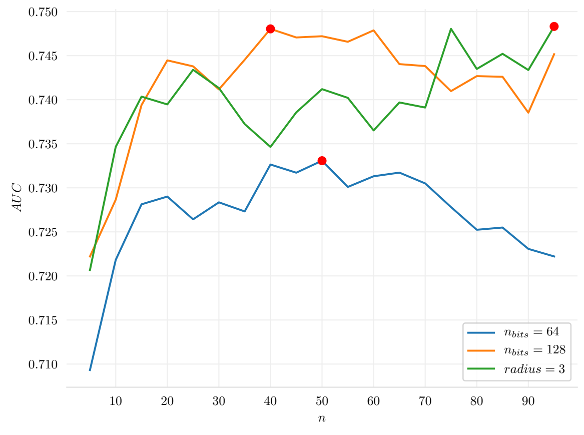
The parameter influencing the score of this method are the number of neighbors considered (`n_neighbors`), the weight associated to each neighbor w.r.t. the distance (`weights`) and the distance metric used (`metric`). For weights, we tried two options : independent of the distance and proportional to the inverse of the distances. For metrics, we tried every bit vector metrics available [6].

As one can see in Figures 1, the uniform weights have the upper hand when fewer (< 40) neighbors are considered but it is the opposite afterwards, yet very slightly. By simplicity, we selected the uniform weights with 20 neighbors which seemed a fine choice.

Concerning the metrics, it seems that none of them is significantly better than the default one. Once more by simplicity, we sticked to the default one.



(a) Uniform weights



(b) Anti-proportional weights

Figure 1 – AUC estimation for KNN w.r.t. the numbers of neighbors.

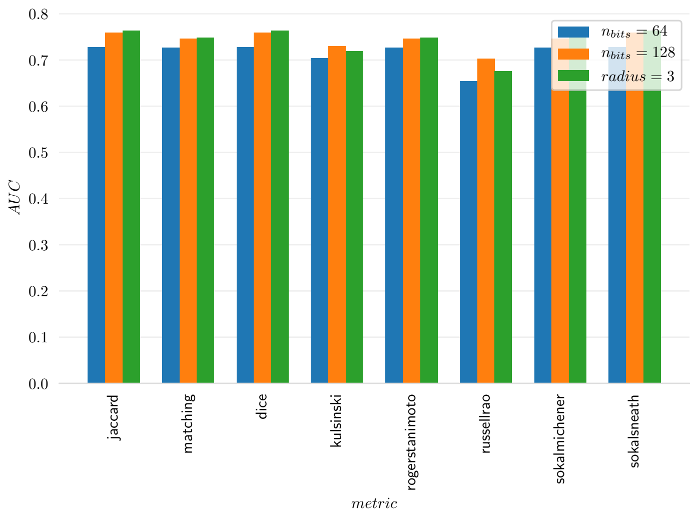


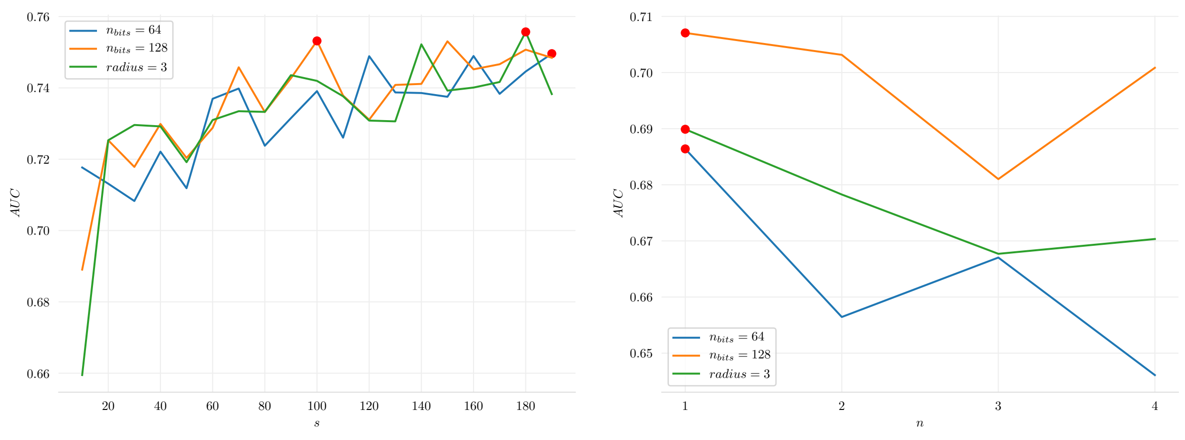
Figure 2 – AUC estimation for KNN w.r.t. the distance metric. (uniform weights and 20 neighbors)

What we can see, however, is that a greater number of bits induces better predictions. As stated above, this observation was made only at the end of the challenge, when we tried other fingerprints (increasing the number of bits with morgan did not provide better results for all models tested). If we had to provide another model, we would obviously increase `nBits` and use another fingerprint, as it would provide better results than the models we have designed.

4.2 Multilayer perceptron

The main elements of this method are the hidden layers shape and the activation function. We experienced a lot with the former and only a few with the latter. Indeed, changing the activation function for `'relu'` (default) to anything else seemed to worsen the results. We therefore varied the number of neurons per layer and the number of layers only.

Concerning the number of layers, it can be seen that the increase in the number of layers



(a) Number of neurons (1 layer) (b) Number of layers (16 neurons per layer)

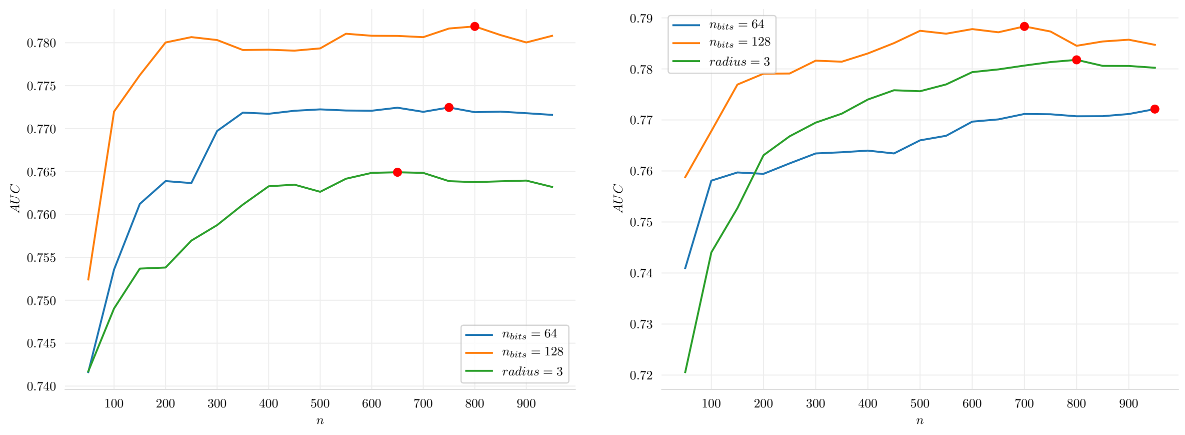
Figure 3 – AUC estimation for MLP w.r.t. the shape of hidden layers.

quickly degrades the score. A single layer seems to provide the best result. This is probably due to overfitting.

Concerning the number of neurons in the first layer, the score increases almost identically with all 3 Morgan fingerprints. A number of neurons between 100 and 200 seems to provide the best results.

4.3 Random forest classifier

The main parameter of this classifier is the number of trees in the forest. We therefore varied it while leaving the other parameters at their default values.



(a) Unbalanced class weights (b) Balanced class weights

Figure 4 – AUC estimation for RFC w.r.t. the number of trees.

In both cases (unbalanced and balanced class weights), the ROC AUC score appears to increase rapidly for a small number of trees and stabilizes when it increases.

In the unbalanced case, the score seems to stabilize at around 400 trees in the forest. There is a marked difference between the different parameters values for the Morgan fingerprint. The best result is, by far, obtained with 128 bits.

In the balanced case, the score stabilize later around 800 trees in the forest. The difference between fingerprint parameters is, in this case, less marked.

In both cases, the best score is obtain with Morgan using 128 bits. The score is slightly better in the balanced class weights case but the difference is very minimal.

4.4 Support vector machine

The main parameter of SVM is the kernel. Different kernels can deliver completely different predictions and each of them isn't suited to any problem. Therefore, we have implemented a few kernels inspired from the literature.

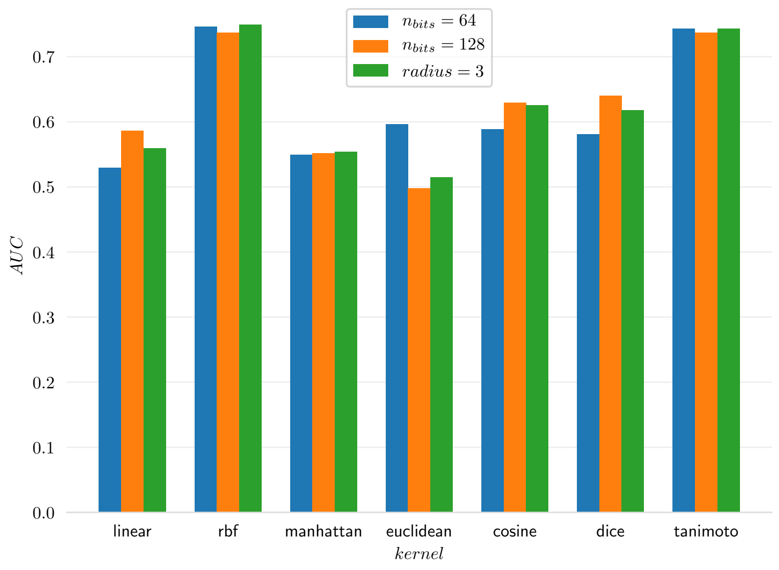


Figure 5 – AUC estimation for SVM w.r.t. the kernel.

As one can see, two kernels are largely above the others : the radial basis function (`rbf`) and the Tanimoto Similarity (`tanimoto`). Since `rbf` has one more tweaking parameter (`gamma`) than `tanimoto`, we chose it.

The other parameter of SVM is C, the regularization parameter. We can clearly see in the Figure 6 that it has no influence on the predicted AUC.

4.5 Ensemble models

After testing some models individually in order to get an idea of their performance and the influence of their parameters on it, we tried to combine the best of them together.

The first way we tried was simply by averaging the probabilities of a set of chosen classifiers. This was done through the `VotingClassifier` of Scikit.

We also tried the `StackingClassifier` of mlxtend. This classifier doesn't average the probabilites, instead it trains another classifier (`meta_classifier`) with the produced probabilities as input.

Finally, we designed our own ensemble classifier `ConsensusClassifier` which, as the two before, takes a list of unfitted classifiers as argument. However, instead of averaging the

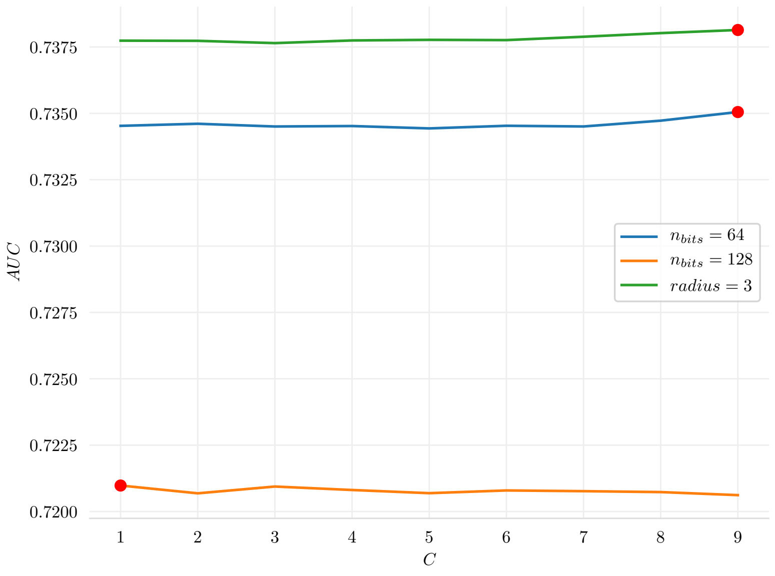


Figure 6 – AUC estimation for SVM w.r.t. C.

probabilities like the `VotingClassifier`, it computes the probability that a majority of classifier will vote "in favour".

Mathematically, we have a set N of n independent Bernoulli random variables $\mathcal{V}_i$ and their success probability p_i. Let x be the number of successful variables $\mathcal{V}_i$. We have,

$$\begin{aligned} P(x \geq j) &= \sum_{k=j}^{n} P(x = k) \\ &= \sum_{k=j}^{n} \left(\sum_{M \in C_N^k} P(\mathcal{V}_i = 1, \forall \mathcal{V}_i \in M) P(\mathcal{V}_i = 0, \forall \mathcal{V}_i \in N \setminus M) \right) \\ &= \sum_{k=j}^{n} \left(\sum_{M \in C_N^k} \prod_{i \in M} p_i \prod_{i \in N \setminus M} (1 - p_i) \right) \end{aligned}$$

where C_N^k represents the set of all k-combinations in N. Therefore, the probability computed by `ConsensusClassifier` is

$$P\left(x \geq \frac{n + (n \bmod 2)}{2}\right) .$$

This model provides better results on our data than the two stated above, which is why this is the one we decided to keep for the tests and the final submission.

4.6 Model Validation Technique

In order to evaluate our models and decide if one was better than another, we decided to rely first on our prediction of the AUC. However, as these were not necessarily reflective of the score obtained on the public leaderboard, that score itself also played a role in our choices.

For example, for the submission `RFC(n_estimators=500)` with *MACCS* fingerprint (cf. Table 1), we obtained an AUC estimation of 0.79 but a public score of 0.71. We therefore decided to not investigate that fingerprint further, although the private score actually was 0.79 as well.

Retrospectively, we could have used other metrics to determine which model was better than another, and not base our decision only on the estimated AUC and the public score. As the public AUC was estimated from only one third of the test set, it could not be very reliable, as the classification problem was unbalanced.

The final models chosen are given at the end of the report.

5 Fingerprints

Feature engineering is the process by which we use knowledge about the problem we face (here, the ability for a chemical compound to inhibit HIV replication) in order to create features that make machine learning efficient. As no member of the group had any knowledge about the domain, we relied on the RDKit library for Python. Given in that library are several fingerprinting methods, which identify and hash topological paths in molecules and use them to set bits in a fingerprints of user-specified lengths (taken from the RDKit documentation).

As explained in the introduction, we tried *Morgan*, *RDKFingerprint*, *Avalon* and *MACCS* fingerprints. Although we started using different fingerprints late in the challenge, we had time to test them on some submissions and found out that, in these tests, *Morgan* provided better results, although the variation in scores was not very high. However, when comparing the scores obtained on the private leaderboard, it was clear that *MACCS* performed better, leading to 0.05 increase in most models in which it had been tested, while the models tested with *Morgan* that performed well on the public leaderboard provided worse results in general in the private one. This may come from the fact that the dataset used for the public leaderboard was not very representative of the whole dataset.

When testing a model (a consensus model containing KNN, MLP, SVM and RFC) with *Avalon* and 512 bits for the fingerprinting, we reached a public score of 0.7564 and a private score of 0.8116. Testing the same submission and only changing *avalon* with *Morgan* gives us a public score of 0.7678 and a private one of 0.7950. We can see that *Morgan* performed better on the "public dataset", all other things being equal, and worse on the private one than *avalon*. As we based our iterative search for the best model partly on the score obtained on the public leaderboard, this is the reason we stuck with *Morgan*, although the estimated AUC was almost 0.02 less for *Morgan*.

6 Performance summary

The table 1 shows the performance of different models tested. Each model was evaluated according to several scores :

- AUC_{pred}, the score that we personally calculated;
- AUC_{publ}, the public score obtained on the Kaggle platform;

- AUC_{priv}, the private score obtained on the Kaggle platform;
- VS, the validation score, calculated as follow :

$$VS = AUC_{priv} - |AUC_{pred} - AUC_{priv}|$$

Fingerprint	Model	AUC			VS
		pred	*publ*	*priv*	
`Morgan(radius=2, nBits=128, useFeatures=False)`	`KNN(n_neighbors=20)`	0.7417	0.7062	0.7494	0.7417
	`MLP(layers_sizes=(100,))`	0.7449	0.7217	0.7512	0.7449
	`SVM(kernel='rbf', gamma='auto', C=1)`	0.7334	0.7027	0.7418	0.7334
	`RFC(n_estimators=500)`	0.7510	0.7415	0.7827	0.7510
`Morgan(radius=2, nBits=128, useFeatures=True)`	`KNN(n_neighbors=20)`	0.7477	0.7469	0.7547	0.7477
	`MLP(layers_sizes=(100,))`	0.7531	0.7342	0.7289	0.7047
	`SVM(kernel='rbf', gamma='auto', C=1)`	0.7209	0.7536	0.7250	0.7209
	`RFC(n_estimators=500)`	0.7793	0.7745	0.7758	0.7723
	`RFC(n_estimators=500, class_weight='balanced')`	0.7874	0.7798	0.7776	0.7678
	`ConsensusClassifier([1])`	0.7813	0.8007	0.7784	0.7755
	`ConsensusClassifier([2])`	0.7829	0.8034	0.7754	0.7679
	`VotingClassifier([1])`	0.7823	0.7953	0.7769	0.7715
	`StackingClassifier([1], meta_classifier= MLP(layers_sizes=(100,)))`	0.7886	0.7825	0.7771	0.7656
	`RFC(n_estimators=100)`[3]	0.7860	0.7936	0.7918	0.7860
`MACCS()`	`KNN(n_neighbors=20)`	0.7468	0.7372	0.7739	0.7468
	`MLP(layers_sizes=(100,))`	0.7384	0.7479	0.7603	0.7384
	`RFC(n_estimators=500)`	0.7918	0.7154	0.7903	0.7888
`Morgan(radius=2, nBits=512, useFeatures=True)`	`ConsensusClassifier([2])`	0.7690	0.7678	0.7950	0.7690
`Avalon(nBits=512)`	`ConsensusClassifier([2])`	0.7875	0.7564	0.8116	0.7875
`Avalon(nBits=128)`	`ConsensusClassifier([2])`	0.7598	0.7281	0.7905	0.7598

Table 1 – Performance summary of the different models (not exhaustive)

6.1 Final models selected

The two models we selected as our "final" models are the one in blue in the Table 1.

1. The first one is a `ConsensusClassifier`, the ensemble classifier we handcrafted. It is has been used with four classifiers : `KNN(n_neighbors=17)`, `MLP(layers_sizes=(100,))`,

[1] `[KNN(n_neighbors=20), MLP(layers_sizes=(100,)), SVM(kernel='rbf', gamma='auto', C=1), RFC(n_estimators=500)]`

[2] `[KNN(n_neighbors=17), MLP(layers_sizes=(100,)), SVM(kernel='rbf', gamma='auto', C=1), RFC(n_estimators=500)]`

[3] That model hasn't been trained with the whole learning set. Indeed, since the learning set was large (enough) but very unbalanced class-wise, we tried to counterbalance it by removing part ($\frac{6}{7}$) of the inactive components.

`SVM(kernel='rbf', gamma='auto', C=1)` and `RFC(n_estimators=500)`. We chose this classifier for two reasons : 1. it was our best public submission and one of the best in our predictions; 2. because we built it ourselves, we were thrilled to see it perform better than other pre-implemented ensemble classifiers. Unfortunately, by comparing the predicted AUC (and the public score) with the private score, we deduce that this model was probably slightly overfitting the public data.

2. The second one is a Random Forest Classifier with 100 estimators. However, as explained before[3], this model hasn't been trained with the whole learning. As a result, this model is less likely to overfit and the computation (fitting) time decreased dramatically. Yet, it was one of our best submission both for the AUC prediction and the public score. We therefore preferred it over slightly better models since it was one of the simplest, certainly the quickest to train and, as the main reason, because it probably overfitted less the public dataset than other models. Indeed, we can see on the final results that it was probably correct as the two scores (public and private) do not differ much.

7 Conclusion

This project allowed us to review and apply many of the concepts seen in class, mainly the use of supervised learning techniques and the evaluation of their performance.

We also had the opportunity to investigate models that were not seen in class, as well as to get information in scientific research papers.

Although our selected models performed quite well on the public leaderboard (with AUC of 0.79360 and 0.80340), we found out that they underperformed on the private one, while some models that we rejected due to their low scores on the public leaderboard showed very good performance on the private one. The fact that our *ConsensusClassifier* underperformed probably comes from the fact that it overfits the data used to estimate the public score. Indeed, some of the models we rejected show a score of more than 0.8, which is better than both models we chose to be evaluated, but a public score of $0.75 - 0.76$.

Were we to select a new model to evaluate, we would also make sure that the number of bits used for fingerprinting is higher than what we used in our submitted models (128 bits), as the higher that number, the more complex the feature that can be detected, and we would also base our selection of algorithms on our predicted AUC without paying much attention to the public score, as our prediction was usually more reliable in that regards (as we realized *a posteriori*). Indeed, some models discarded early due to their low public score had a high predicted score of more than 0.79 and provided more than 0.80 on the private leaderboard (simple Random Forest Classifiers with 500 or 1000 trees for example), although they showed a public score of $0.75 - 0.76$.

A Code organization

Our code consists of 6 files :

- `chart.py` : this file contains customised plots functions in order to best present our results.
- `fingerprints.py` : this file contains several functions allowing us to test different fingerprints.
- `kernels.py` : this file contains several personal kernel implementations.
- `main.py` : this file is the main file of our code. It retrieves the data from the training set, instantiates and trains the models, evaluates their performance and creates the submission file.
- `models.py` : this file contains the implementation of all our models. Each model is represented by a class.
- `utils.py` : this file contains the various functions of our code. These functions, allowing to read CSV files, convert SMILES representations into fingerprints and create the submission file, have been taken from the toy submission proposed on the Kaggle platform.

References

[1] *IML2019 - Activity prediction for chemical compounds.* 2019. URL: https://www.kaggle.com/c/iml2019/.

[2] *Variance Threshold.* 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html.

[3] *ROC AUC Metric.* 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html.

[4] *ShuffleSplit cross-validator.* 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ShuffleSplit.html#sklearn.model_selection.ShuffleSplit.

[5] *Feature Definitions Used in the Morgan Fingerprints.* 2019. URL: https://www.rdkit.org/docs/GettingStartedInPython.html#feature-definitions-used-in-the-morgan-fingerprints.

[6] *Scikit Distance Metrics.* 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html#sklearn.neighbors.DistanceMetric.

[7] Yang Li, Yujia Tian, Zijian Qin, and Aixia Yan. "Classification of HIV-1 Protease Inhibitors by Machine Learning Methods". In: *ACS omega* 3.11 (2018), pp. 15837–15849.

[8] Samina Kausar and Andre O Falcao. "Analysis and Comparison of Vector Space and Metric Space Representations in QSAR Modeling". In: *Molecules* 24.9 (2019), p. 1698.

[9] Sinyoung Kim and Kwang-Hwi Cho. "PyQSAR: A Fast QSAR Modeling Platform Using Machine Learning and Jupyter Notebook". In: *Bulletin of the Korean Chemical Society* 40.1 (2019), pp. 39–44.

[10] Jean-Pierre Doucet, Florent Barbault, Hairong Xia, Annick Panaye, and Botao Fan. "Nonlinear SVM approaches to QSPR/QSAR studies and drug design". In: *Current Computer-Aided Drug Design* 3.4 (2007), pp. 263–289.

[11] Aziz Yasri and David Hartsough. "Toward an optimal procedure for variable selection and QSAR model building". In: *Journal of chemical information and computer sciences* 41.5 (2001), pp. 1218–1227.

[12] Stefan Kramer, Luc De Raedt, and Christoph Helma. "Molecular feature mining in HIV data". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM. 2001, pp. 136–143.