DNA Binding Sites Prediction

Federico Betti, Ioannis Mavrothalassitis, Luca Rossi
CS-433 Machine Learning, EPFL Lausanne, Switzerland

***Abstract*—In this paper we present our approach to the problem of predicting the probability that a given DNA sequence (from which features are extracted by domain experts) is a binding site, i.e. the so called RelKa value. This is both a regression and classification problem on an extremely imbalanced dataset: one is interested both in predicting well the RelKa value ranging in $[0, 1]$ and to classify accurately in $\{0, 1\}$ DNA sequences that are respectively not likely or likely to be binding sites according to some accurately defined threshold.**

I. Introduction

Interactions between proteins and DNA play an important role in many essential biological processes such as DNA replication, transcription, splicing, and repair. The identification of amino acid residues involved in DNA-binding sites is critical for understanding the mechanism of these biological activities. The main characteristic of the dataset is that it is **highly imbalanced** towards RelKa values which are below 0.7, i.e. the majority of the samples are not likely to be binding sites. On the other hand **we aim at predicting and classifying well the minority class**, namely the samples with high RelKa. Indeed, specific genetic tests will be carried out on this datapoints, so it is crucial to build a model which can classify and predict well the minority class and not only have a high accuracy or a small loss overall [3]. While for the classification task our main focus is on improving as much as possible the accuracy on the minority class (while a controlled increase of false positives may not be so concerning), for the regression task we aim at having a uniform distribution of the error for all intervals of RelKa.

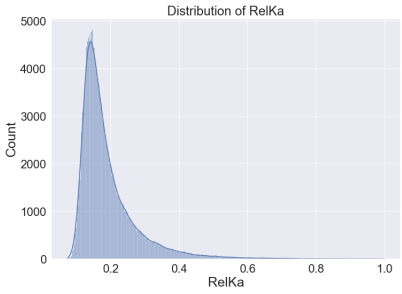

Fig. 1: Distribution of the RelKa value: only 0.2% of the samples have a value above 0.7

We are given in particular a dataset with 170580 training samples and 321 features. The first column (initially dropped [1]) represents the DNA sequence and the other 320 features are mechanical properties coming from a cgdna+ model. The labels in the last column represent the RelKa value which we want to predict and classify as well as possible.

II. Feature Pre-Processing

In order not to fall in the well-known *accuracy paradox*, we first perform a guided split of the dataset between training data and testing data. We want in particular the minority class to be represented well enough in the testing set to test the accuracy of our model on the samples with RelKa in the range of interest. To this aim, we first choose an *importance boundary* which, given the graphical evidence in Fig.1, is set to $b = 0.7$, dividing the whole dataset depending on whether the RelKa of each sample is below or above such a threshold. We then proceed to randomly split each of the two classes in training and test folds with a proportion $70\% - 30\%$ to remain coherent with the above. In the end we stack the training and test folds coming from the two classes, shuffling them to ensure a random distribution of the minority class. For the classification task we coherently choose the threshold to also be $b = 0.7$ allowing us to have both an high enough probability for the DNA sequence to be a binding site in case of positive prediction and enough data to train on.

Outliers detection

In classical fashion of feature pre-processing we look for outliers samples in the training data by performing an Isolation Forest Algorithm.

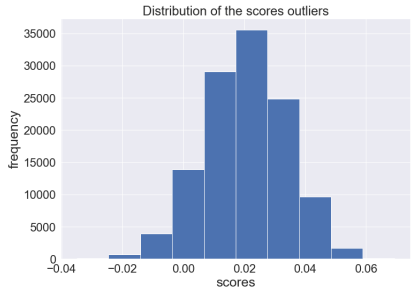

Fig. 2: Isolation Forest Algorithm scores distribution as returned by the decision function

We recall that the standard Isolation Forest Algorithm returns for each sample a real value in the range $[-\frac{1}{2}, \frac{1}{2}]$, considering then as outliers the samples whose score is negative[2]. We see from the plot above that for our dataset, on the other hand, there is a high percentage of scores around 0. Moreover, the outliers found by the algorithm correspond to RelKa value which are not necessarily high. This may suggest looking for

[1] later restored for the one hot encoding technique tried in the end

[2] changing this decision boundary can also be tried but the results are similar

another outliers detection algorithm. However, as we expect for high dimensionality and highly imbalanced datasets, decision-trees based algorithm for outliers detection are outperforming the other algorithms tried (One-Class SVM, Hidden Markov models) especially in terms of computational efficiency. As better explained in the Results section, dropping the outliers improves our results both in regression and in classification (for the latter, in particular on the minority class due to implicit undersampling).

Undersampling and oversampling

Indeed, it is a well known fact (see [1]) that for highly imbalanced datasets the joint effect of ad-hoc undersampling and oversampling may lead to high increases of the performance. Concerning the undersampling strategy, again **the high dimensionality of the dataset makes standard density-based clustering algorithms for undersampling** such as k-means useless or at least **not optimal for our purposes**. We want in particular to reduce properly the peak of our distribution of RelKa (see Figure 1) without changing drastically the distribution of the dataset. A first reasonable idea may be to perform a trivial **random undersampling** algorithm on the samples of the majority class (with RelKa below b) to reduce the relative percentage of the latter in the training set. We present here another interesting algorithm which is a suitable undersampling technique and which gave nice results for our problem: in particular, we implement the idea of **Particle Stacking Undersampling** algorithm suggested in [2].

Algorithm 1 PSU Undersampling

Input: The majority class datapoints $\{X_1, \dots, X_M\}$ and the undersampling rate r_u.

1) Calculate the centroid $C = \frac{1}{M}\sum_{i=1}^{M} X_i$ of the majority class.
2) Calculate the 2-distance between each majority class sample X_i and the centroid: $D_2 = \{D_{2,i} = ||C - X_i||_2\}_{i=1}^{M}$.
3) Sort D_2 in $p = \lceil r_u M \rceil$ partitions $[s_1, \dots, s_p]$.
4) Set $\hat{X}_1$ to be the datapoint corresponding to the last distance in s_1.
5) **For** k = 2, …, p
   Set $\hat{X}_k$ to be the furthest data point X_j with $D_{2,j}$ in s_k with respect to the resampled dataset $[\hat{X}_1, \dots, \hat{X}_{k-1}]$.

Output: The resampled dataset $\{\hat{X}_1, \dots, \hat{X}_p\}$

We applied in particular the standard version of the PSU Undersampling to the classification problem.

For the regression part our goal to have a uniform distribution of the error suggests that applying these algorithms separately on each interval of the label RelKa may work very well (see for example [4] or [6]). That is, we use **stratification** in a grid of intervals, then applying our algorithm only on the samples whose label falls in a particular stratum in a proper way.

Algorithm 2 Random Stratified Undersampling

Input: The majority class M, the width of the intervals in the grid w, the importance boundary b.

1) Divide the interval $[0, b]$ in $\frac{b}{w}$ intervals $\{I_i\}_i$.
2) Extrapolate from M only the samples with RelKa $\in I_i$ $S_i = \{X_{i,j}\}_j$ such that $k_i = |S_i|$.
3) Drop any sample in S_i with probability f_i, where $f_i = k_i/|M|$.

Output: The stacked remaining samples from each stratum

Note that in Algorithm 2 in expectation we keep only $(1-f_i)k_i$ samples in the i-th strata, so the higher the frequency of an interval the more we are likely to drop samples there. We use Algorithm 2 for the undersampling in the regression task.

To perform oversampling, we cannot rely on the standard version of SMOTE algorithm as the latter may suffer from the curse of dimensionality due to the large size of the dataset. We thus apply a **variant of the SMOTE oversampling technique**: we present here the main features of this algorithm, referring to [5] for more details.

Algorithm 3 Variant of SMOTE Oversampling

Input: The already undersampled training dataset D, the final proportion between the minority and the majority class r_o, the attribute number s, the number of nearest neighbours k to be considered for each sample, the exponent q of the Minkowsky distance, the importance boundary b (0.7 by default).

1) Split D into the set M of majority class samples and the set m of minority class samples according to b.
2) For each i-th feature, calculate the associated **Fischer score**[3]
$$FS_i = \frac{\mu_{i,m} - \mu_{i,M}}{\sigma_{i,m} + \sigma_{i,M}}, \tag{1}$$
where $\mu_{i,m}$ and $\sigma_{i,m}$ represent the i-th feature's mean and standard deviation for the datapoints in the minority class, and similarly for the majority class M.
3) Compute the q-Minkowsky distance between the samples in the dataset considering only the s highest scored features according to (1).
4) Find the k nearest neighbours $\{x_{i,j}\}_{j=1}^{k}$ of each sample x_i in the minority class.
5) **For** i = 1, ..., $N = \lceil r_o|M| - |m| \rceil$
   Let $\lambda_i \sim Unif([0, 1])$. Randomly pick a sample $x_i \in m$ with label y_i and one of its k nearest neighbours $x_{i,j}$ with label $y_{i,j}$ for some $j = 1, \dots, k$. Generate a new sample $\tilde{x_i}$ with label $\tilde{y_i}$ by means of
$$\tilde{x_i} = x_i + \lambda_i(x_{i,j} - x_i), \quad \tilde{y_i} = y_i + \lambda_i(y_{i,j} - y_i). \tag{2}$$

Output: The new created samples $\{\tilde{x_i}\}_{i=1}^{N}$

[3]also other correlation scores such as the Spearman factor were tried but the results were similar

Algorithm 3 is applied both in the regression and in the classification problems as it is outperforming the other oversampling algorithms tried.
More in general, many other combinations of undersampling and oversampling techniques can be tried. We present for the interested reader the stratified versions of Algorithms 1 and 3 in the appendix, together with a slight modification of 1 which was performing better than the standard algorithm for the classification problem.

Boxcox Transformation

Another fact that one can notice is that the distribution of the features is not at all non-skewed and well behaving; this leads us to apply a **Box-Cox Transformation**, i.e. a power transform that assumes the values of the input variable to be strictly positive and transforms non-normal dependent variables into a normal shape.

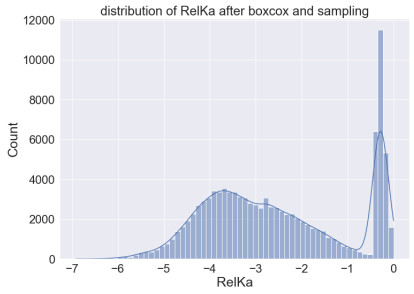

Fig. 3: Distribution of the labels in the resampled training set after Box-Cox Transformation

Note that having the labels roughly distributed according to a Gaussian enables us to make better predictions on the minority class. Clearly this is only applied to the regression problem.

III. Model selection and validation

Regression model

For the regression task, we choose to train the model using **XGBoost Regressor** which is in principle known to be quite robust for imbalanced datasets. Moreover it can be adapted quite easily to improve the performance on the minority class. We thus perform **5-fold cross validation** for tuning the (randomly picked in a grid) hyperparameters of the regressor model[4]. In this case tuning the hyperparameters is just slightly improving the performance (or not at all), synthom of the **reliability and robustness of our model** whose score is not relying too much on the chosen hyperparameters. The only difference with respect to the classification case is a higher computational effort to perform the training of our model.

Classification model

For the classification task, we choose to train the model using decision-trees based classification algorithms. Note that even though they may be biased towards the majority class, we are training now our model on a reasonably balanced dataset so this may not represent a problem anymore. On the other hand these type of algorithms are computationally less expensive for high-dimensionality training purposes like this one. In particular, the latter reason leads us to choose a **Random Forest Classifier** on which we perform **5-fold cross-validation** for the tuning of its main (randomly picked in a grid) parameters (e.g. the number of estimators, the minimum number of samples required to split an internal node, the minimum number of samples required to be a leaf node). In our case, augmenting majorly the number of estimators in the forest is not increasing the accuracy on the minority class and is just making heavier the computation.

[4]for the best parameters of both regression and classification models we refer to the README in the GitHub repository attached in the appendix

IV. Results

As explained in the first section, for both regression and classification removing the outliers before any other feature pre-processing of the dataset is improving the performance of our models, and in particular on the minority class. Moreover the undersampling and oversampling strategies adopted are making our model more reliable in the intervals of interest, making huge improvements with respect to the initial scenario presented here.

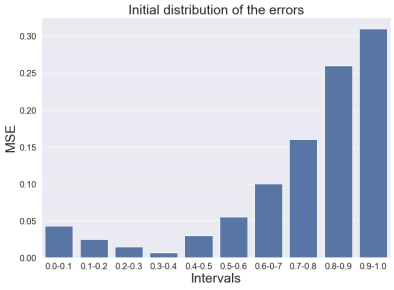

Fig. 4: Initial error distribution

		Predicted	
		0	1
Actual	0	51068	9
	1	60	38

TABLE I: Confusion matrix without any resampling

Let us now discuss separately the final pipelines used for the two tasks and the obtained results.

Regression task

For the regression task we perform **Random Stratified Undersampling** according to Algorithm 2 and **SMOTE Oversampling** according to Algorithm 3 as this combination is outperforming the other algorithms described above, helping us in our objective to make the error distribution as small and uniform as possible over the intervals.

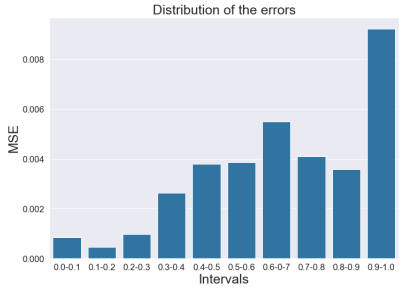

Fig. 5: Error distribution with Random Stratified Undersampling and SMOTE Oversampling

We remark however that also the stratified versions of SMOTE Oversampling and PSU Undersampling combined were working very well giving us an enormous decrease in the error on the last intervals. In particular, in Algorithm 3 we choose to **oversample at a rate $\mathbf{r_o = 0.3}$** [5]. An important fact to be noticed is that we improve a lot on the minority class and on the central intervals by the techniques explained above without reducing at all the "near perfection" of the model in predicting the values of RelKa below 0.2, even after undersampling majorly in that intervals. Obviously, at the beginning, our aim is to find a trade-off between the will of remaining very precise on the majority class and the need of improving on the minority class: Figure 5 shows indeed that we **improve on the area of interest without losing anything** in the "already well controlled" first intervals.

Classification task

For the classification task we perform **PSU undersampling** with **undersampling rate $\mathbf{r_u = 0.1}$** and **SMOTE Oversampling** with **oversampling rate $\mathbf{r_o = 0.3}$**.

		Predicted	
		0	1
Actual	0	50837	240
	1	8	90

TABLE II: Confusion matrix with PSU Undersampling and SMOTE Oversampling

While we can see that in this case the undersampling applied on the majority class leads to a slight decrease in the performances on the latter, at the same time we improve from a 40% to a 93% accuracy on the minority class, so we may consider this a good trade-off and a very good result. The stratified versions of the used algorithms were not performing well for this classification task as they were only giving around $60 - 70\%$ accuracy on the minority class.

Error Analysis

In order to further improve our performance, we try to perform some error analysis to see if there is something reminiscent of a cluster of badly predicted samples or if some common values of a feature could be associated to an error in the prediction. Obviously for the classification task the errors are the elements which fall on the counterdiagonal of the confusion matrix, while for the regression we choose to consider as errors predictions of the RelKa value which are differing from the true label more than 0.15. After extracting the errors committed in our best runs in both tasks, we add an extra label $\hat{y} \in \{0, 1\}$ (where 1 denotes an error on the training set), hoping to get a cleaner view of the situation. First of all, we try to plot the distribution of the features for the correct samples and the errors in the same plot. Unfortunately the two distribution look very similar and no further conclusion can be reached. Secondly, we use an **autoencoder** to project the features on a two dimensional space to see if the errors are kind of separable from the correct classifications and predictions. If this is the case, we could consider using a clustering algorithm to isolate the errors, picking then a more suitable model for that particular cluster. Unfortunately, also in the second case no cluster of misclassified samples can be recognized.

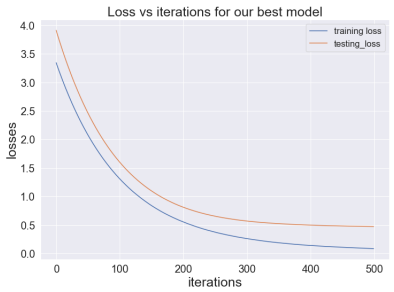

Fig. 6: Train and test loss from the regression model against the number of iterations

As we see from the plot below we are not overfitting the data. We may explain this by saying that even though we are using a relatively complex model, the pre-processing pipeline is not making the model overfit.

V. Discussion and final remarks

To summarize we succeed in both tasks of predicting and classifying well the RelKa values above the selected threshold without worsening majorly the results on the majority class and with an almost uniform distribution of the error in the regression part. Note that this was done in a data centric approach, picking a baseline model to start and trying then to improve the performance without changing it. An implicit advantage of the undersampling performed was an exponential fastening of the training of the model; moreover on the balanced dataset we can use standard accuracy measures without introducing penalized ones in favour of the minority class. Another two things that were tried in the end were a one hot and a sequential encoding for the DNA sequence to be added to the existing features, but surprisingly this was not increasing the performance in both the tasks. This may be due to the fact that adding those extra features is giving indeed too much information on the samples and hence we may overfit the data.

[5] this parameter is chosen by a separate grid search procedure on a validation set optimizing with respect to the overall accuracy / MSE

VI. Appendix

Algorithm 4 Variant of PSU Undersampling

Input: The majority class datapoints $\{X_1, \dots, X_M\}$ and the undersampling rate r_u.

1) Calculate the centroid $C = \frac{1}{M}\sum_{i=1}^{M} X_i$ of the majority class.
2) Calculate the 2-distance between each majority class sample X_i and the centroid: $D_2 = \{D_{2,i} = ||C - X_i||_2\}_{i=1}^{M}$.
3) Sort D_2 in $p = \lceil r_u M \rceil$ partitions $[s_1, \dots, s_p]$.
4) **For** k = 1, ..., p
   Set
$$\hat{X}_k = \underset{X_i : D_{2,i} \in s_k}{\operatorname{argmin}} \min_{X_j \in m} ||X_i - X_j||_2 \tag{3}$$
   where m is the set of all datapoints in the minority class.

Output: The resampled dataset $\{\hat{X}_1, \dots, \hat{X}_p\}$

Algorithm 5 Stratified SMOTE Oversampling

Input: Same inputs of Algorithm 3, in addition the width of the intervals in the grid w (0.05 by default), the importance boundary b (0.7 by default)

1) Apply steps $1-3$ of the standard SMOTE Oversampling (we use the same notation as above)
2) Divide the interval $[b, 1]$ in $\frac{1-b}{w}$ intervals $\{I_i\}_i$.
   For each stratum I_i:
3) Extrapolate from the minority class m only the samples with RelKa $\in I_i$ $S_i = \{x_{i,j}\}_{j=1}^{k_i=|S_i|}$.
4) Apply steps $4-5$ of Algorithm (3) to S_i with
$$N = N_i = (r_o|M| - |m|)|\frac{(1-f_i)}{\sum_j 1 - f_j},$$
   where $f_i = k_i/|m|$, and $\lambda_j \sim Unif([0, f_i])$[6]. Thus, generate N_i new samples $\{\tilde{x}_{i,j}\}_{j=1}^{N_i}$.

Output: The new created samples $\{\tilde{x}_{i,j}\}_{j=1}^{N_i}$ for $i = 1, \dots, \frac{1-b}{w}$

Algorithm 6 Stratified PSU Undersampling

Input: The majority class datapoints $\{X_1, \dots, X_M\}$, the width of the intervals in the grid w (0.05 by default), the importance boundary b (0.7 by default)

1) Apply steps $1-2$ of Algorithm 1
2) Divide the interval $[0, b]$ in $\frac{b}{w}$ intervals $\{I_i\}_i$.
   For each stratum I_i:
3) Extrapolate from M only the samples with RelKa $\in I_i$ $S_i = \{X_{i,j}\}_{j=1}^{k_i=|S_i|}$.
4) Apply steps $4-5$ of Algorithm 1 to the samples in S_i using either (3) or the standard version with a number of partitions $p = p_i = k_i(1-f_i)$, f_i being $k_i/|M|$ [7].

Output: The stacked remaining samples from each stratum

[6] By choosing λ_j like this, if the frequency of a strata is lower the algorithm is likely to create according to equation (2) a new sample with label in the under-represented strata

[7] Note that by construction with respect to Algorithm 1 we don't have anymore an overall undersampling rate but we simply fix the number of partitions for the distances in each strata

References

[1] Evgeny Burnaev, Pavel Erofeev, and Artem Papanov. "Influence of resampling on accuracy of imbalanced classification". In: *Eighth international conference on machine vision (ICMV 2015)*. Vol. 9875. International Society for Optics and Photonics. 2015, p. 987521.

[2] Yong-Seok Jeon and Dong-Joon Lim. "PSU: Particle stacking undersampling method for highly imbalanced big data". In: *IEEE Access* 8 (2020), pp. 131920–131927.

[3] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. "Handling imbalanced datasets: A review". In: *GESTS International Transactions on Computer Science and Engineering* 30.1 (2006), pp. 25–36.

[4] Shoushan Li et al. "Imbalanced sentiment classification". In: *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2011, pp. 2469–2472.

[5] Sebastián Maldonado, Julio López, and Carla Vairetti. "An alternative SMOTE oversampling strategy for high-dimensional datasets". In: *Applied Soft Computing* 76 (2019), pp. 380–389.

[6] Iman Nekooeimehr and Susana K Lai-Yuen. "Adaptive semi-unsupervised weighted oversampling (A-SUWO) for imbalanced datasets". In: *Expert Systems with Applications* 46 (2016), pp. 405–416.