Genome-wide prediction of cis-regulatory regions using Random Forest

Nicola Amadio

AY 2019/2020

Abstract

In this work we focused on predicting cis-regulatory regions (CRR) using a Machine Learning based approach and in particular the Random Forest predictor. The problem of identifying active CRRs in the human genome is critical for understanding gene regulation and assessing the impact of genetic variation on phenotype. Moreover, the predicted annotations of CRRs will provide broad utility for genome interpretation from functional genomics to clinical applications. The use of computational methods to detect the locations of promoters and enhancers has been a key focus of bioinformatics for twenty years, with a growing interest in Machine Learning based approaches in the last five. For this work we took as an important reference the work done by [1], where they conducted a very comprehensive analysis. This paper is structured as follows: introduction, analysis of the dataset, explanation of the algorithm, description of the experiments and finally a comment on the experiments results. Our code has been open-sourced and it's available at [5], in order to make it easy to reproduce experiments and results.

1 Introduction

We worked on two different problems.

1.1 A-E versus A-P

This task consists of building a classifier that correctly distinguishes between active enhancers and promoters (A-E versus A-P). It turns out that A-E and A-P are highly separable and a Random Forest classifier achieves good results with an out-of-the-box model, and slightly better results after some hyperparameter tuning.

1.2 A-E and A-P versus A-X, I-E, I-P, I-X, and UK

The second problem is the one of distinguishing A-E and A-P from the rest of the classes (A-X, I-E, I-P, I-X, and UK). This task is learned with the same accuracy as the first one with the same Random Forest model. It only takes more computational time because of the bigger dataset.

2 Description of the datasets

We used the precisely annotated [2] promoters and enhancers, which provide the largest experimentally defined collection of CRRs. We directly used the form provided by [4], which is organized as follows: four cell types (GM12878, HelaS3, HepG2, and K562), each of which has a 72-135 features dataset (epigenomic-data) and a 7 classes label set (A-E, A-P, A-X, I-E, I-P, I-X, and UK). We didn't use the sequences data in this work, although it would be interesting to try exploiting those as well. We tried running our experiments over all cell lines realizing that the results were quite consistent over them. We used some functions from the Pandas' library [3] to process our data.

2.1 A-E versus A-P

For this task we filtered our data set only caring about the cells of type A-E and A-P, with the result of having a much smaller table (approximately by a factor of 10). We used a mask vector for this filtering action.

2.2 A-E and A-P versus A-X, I-E, I-P, I-X, and UK

Here we used the entire dataset. We just labeled it into two macro-classes: one characterized by A-E and A-P and the other by the remaining classes.

3 Explanation of the algorithm

3.1 Tree Predictor

A tree predictor [6] $h_T : X \mapsto Y$ is a predictor defined by a rooted and ordered tree T whose internal nodes are tagged with tests and whose leaves are tagged with labels in Y. A test for an internal node with k children is a function $f : X_i \mapsto 1, ..., k$. The function f maps each element of X into a node children. The prediction $h_T(x)$ is computed by testing x with the test at the root and by traversing the tree until one of its leaves is reached, by evaluating the tests present at each node in the path and choosing the next node accordingly. We now describe a generic method to construct a binary tree given a training set S.

1. Initialization: Create T with only the root l and let $S_l = S$. Let the label associated with the root be the most frequent label in S_l.

2. Main loop: pick a leaf l and replace it with an internal node v creating two children l' (first child) and l'' (second child). Pick an attribute i and a test $f : X_i \mapsto \{1, 2\}$. Associate the test f with v and partition S_l in the two subsets $S_{l'} = \{(x_t, y_t) \in S_l : f(x_t, i) = 1\}$ and $S_{l''} = \{(x_t, y_t) \in S_l : f(x_t, i) = 2\}$. Let

the labels associated with l' and l'' be, respectively, the most frequent labels in $S_{l'}$ and $S_{l''}$. Tree predictors, as any other model, may suffer from overfitting. In this case the relevant parameter is the number of tree nodes or, in other words, the maximal depth the tree is allowed to reach. If the number of tree nodes grows too much compared to the cardinality of the training set, then the tree may overfit the training data. For this reason, the choice of the leaf to expand should approximately guarantee the largest decrease in the training error. In our case, the function used to evaluate the training error decrease was the Gini Function, which is the default used by the Scikit-learn Tree Predictor. It's important to point out that Decision Trees can be both parametric or non-parametric, depending on how the maximal depth is decided: if it's decided beforehand, then we're dealing with a parametric algorithm, if we decide it on run-time (either using the Gini Function or any other information-theoretic coefficient), then it's a non-parametric one.

3.2 Random Forest

A Random Forest algorithm [6] averages multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

4 Description of the Experiments

We went through a very standard type of pipeline that's usually used in the Machine Learning community. First we used Pandas for importing the data set and then transformed the data frames into Numpy arrays. After that, we adjusted our data set differently for the two problems: for the first one, we removed the rows which were labeled differently than "A-P" and "A-E", and, for the second, we made the labeling binary by grouping the labels "A-P" and "A-E" into one class and all the others into another. Resampling the data improved overall accuracy, especially for the second problem and in particular the Precision and Recall coefficients. We used the SMOTE [7] algorithm. We then split the data into training and test sets; varying the percentage for the split from 0.2 to 0.4 for the test set didn't really make a big difference when looking at the experimental results. For the training, we used the models and algorithms provided by scikit-learn. We tried tuning the hyperparameters with a Grid Search but the results didn't improve much. We finally went on measuring different kinds of parameters, which we are going to dive into in the next paragraph.

5 Experiments Results

Both problems were learned with satisfying results, over every coefficient we computed. We were particularly interested in the Test Accuracy (performance of our classifier over the test set), Precision ((number of True Positives)/(number of True Positives plus number of False Positives)) and Recall ((number of True Positives)/(number of True Positives plus number of False Negatives)).

We then displayed the PRC Curve (Precision over Recall curve) along with its Average Precision (AP) coefficient, and the ROC Curve (Recall over False Positive Rate, where False Positive Rate is the number of False Positives over the sum of False Positives and True Negatives), along with its Area Under the Curve (AUC) measure.

It's worth pointing out for the sake of clarity that "Positive" and "Negative" in our experiments meant respectively "First Class" and "Second Class" of our binary classification.

For the first problem, i.e. A-E versus A-P, we had:

Test Accuracy of over 0.96 with a Train Accuracy of around 0.99, so we can say we didn't overfit too much. Precision and Recall both around 0.962, so we can say we were equally precise in predicting one class or the other.

For the second problem, i.e. A-E and A-P versus A-X, I-E, I-P, I-X, and UK, we had:

Test Accuracy of over 0.97 with a Train Accuracy of around 0.99, so we we didn't overfit in this case either.

Precision and Recall both around 0.974, therefore we were equally precise in predicting both classes in this case as well.

Finally, let's have a look at the curves we displayed for both problems:

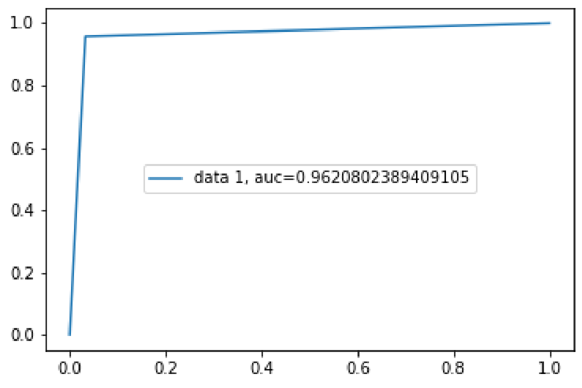

Figure 1: ROC Curve for A-E versus A-P

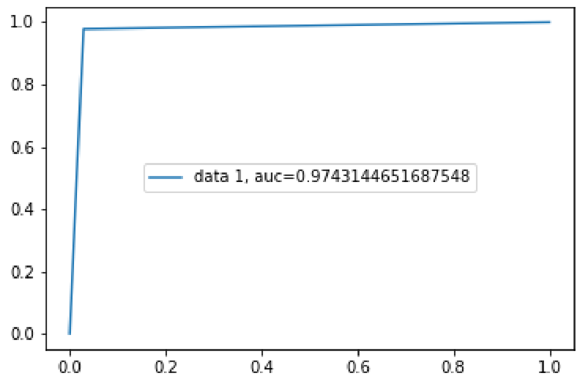

Figure 2: ROC Curve for A-E and A-P versus A-X, I-E, I-P, I-X, and UK

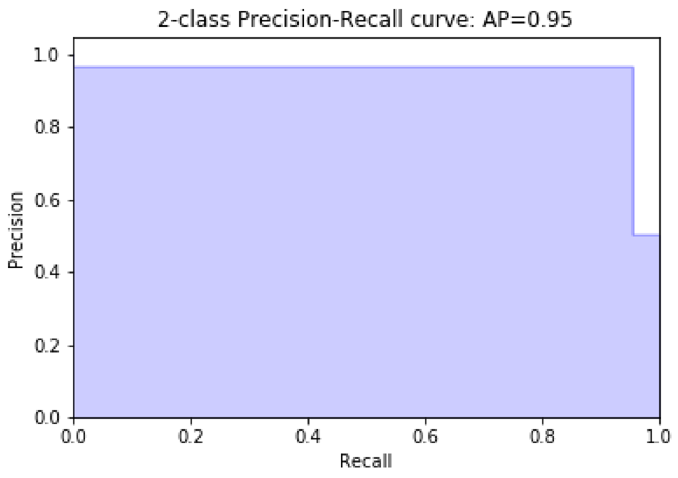

Figure 3: PRC Curve for A-E versus A-P

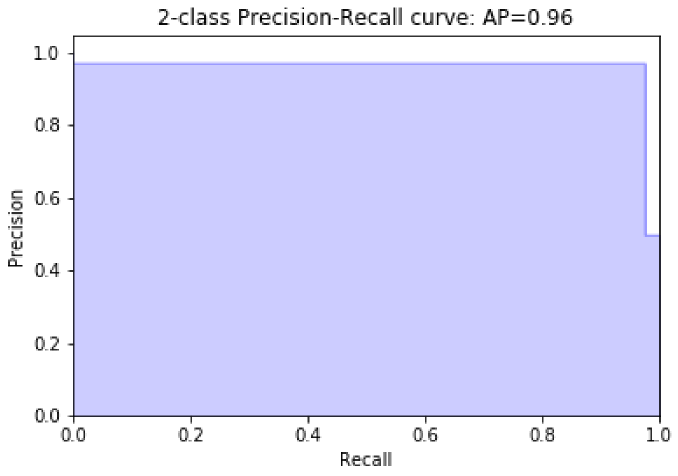

Figure 4: PRC Curve for A-E and A-P versus A-X, I-E, I-P, I-X, and UK

References

[1] Li et al. *Genome-wide prediction of cis-regulatory regions using supervised deep learning methods.* In BMC Bioinformatics, 2018.

[2] Hideya Kawaji, Takeya Kasukawa, Alistair Forrest, Piero Carninci, Yoshihide Hayashizaki *The FANTOM5 collection, a data series underpinning mammalian transcriptome atlases in diverse cell types.* In Scientific Data, 2017.

[3] `https://pandas.pydata.org/`

[4] `https://homes.di.unimi.it/valenti/DATA/ProgettoBioinf1819/`

[5] `https://github.com/amadionix/bioinformatics_project`

[6] Shalev-Shwartz, Shai; Ben-David, Shai. *Understanding Machine Learning. cap. 18.* In Cambridge University Press, 2014.

[7] Bowyer, Kevin W. et al. *SMOTE: Synthetic Minority Over-sampling Technique.* In Journal of Artificial Intelligence Research, 2002.