Predicting Pipeline Failure

Thomas Bordier
MVA
ENS Cachan/Ecole Polytechnique
thomas.bordier@polytechnique.edu

Christophe Lanternier
MVA
ENS Cachan/Ecole Polytechnique
christophe.lanternier@polytechnique.edu

Abstract

This project was realized in the frame of the class "Sparse wavelet representations and classification" taught by Stéphane Mallat, during the fall 2016 semester of the MSc "MVA" at ENS-Cachan.

1 Introduction

The project was led in the scope of a data challenge given by Veolia Research & Innovation. Veolia is in charge of managing fluid related networks such as water distribution. In order to provide the best service possible, Veolia seeks to carry out an efficient maintenance scheme. In order to do so, being able to predict pipeline failure will allow to dispatch maintenance teams in a more efficient way.

2 Algorithm's scoring

As a reminder, the score computed by the competition is the following:

$$Score = 0.6 \times AUC(2014) + 0.4 \times AUC(2015)$$

As we made our way through the challenge, it turned out that a big problem was that the score we computed on our train/test split was completely different from the score announced by the website after submission. This was a major issue, as we were working blind, not knowing if we were really performing better, or simply over-fitting our data a bit more.
To solve this problem, we implemented a way to split our train/test randomly, while making sure that proportions between both classes (pipe with failure and pipe without failure) were still the same as the original train set provided. We then trained a single model on 20 different splits, observing the mean and variance of the series of scores obtained. If variance was small, we could be confident that our model was less likely to over-fit, and trust the mean score to be close to the website's score. This precaution allowed us to set a safe grading environment, and leaded to a big improvement of 0.2 in overall performance.

3 Data representation

	Feature1	Feature2	Feature3	Feature4	Length	Year Construction	YearLast Failure Observed
1	T	IAB	-0.209841	C	3.5972	2001	NaN
2	T	U	2.184992	M	4.0547	1965	NaN

Table 1: Sample from the raw dataset

Above, a sample of the dataset provided by Veolia. One can see that half the features do not allow for interpretation of the values, and thus we cannot use field specific feature engineering. The basic preprocessing step was to change **Feature1**, **Feature2**, **Feature4** into boolean values and to normalize the float valued features.

	Feature3	Length	Year Constr.	YearLast Failure Obs.	P	T	IAB	O	U	C	D	Dr	M
1	-0.001506	0.000494	0.003946	-0.007229	0	1	1	0	0	1	0	0	0
2	0.015677	0.000557	0.014094	-0.007229	0	1	0	0	1	0	0	0	1

Table 2: Basic preprocessing

3.1 Data augmentation

The biggest challenge of this problem is the heavily imbalanced repartition of the dataset. Indeed, in the given training set, only 87 cases of failures are reported, on a total of 19427 pipelines. A classical approach on this challenge would typically lead to heavy over-fitting, or a simple and pure ignorance of the under-represented class. To tackle this problem, we decided to explore ways of over-sampling the under-represented class in order to reestablish balance between both classes.

3.1.1 Basic over-sampling

In our first attempt, we simply decided to duplicate samples of failures, until balance was established. We trained our first models using this simple technique, yielding acceptable performances.
This technique's main downside is that it doesn't create any diversity in the train set, the model still trains on the same few examples, which makes it more likely to over-fit.

3.1.2 SMOTE: Synthetic Minority Over-sampling Technique

This technique was introduced by paper [1]. The core idea is to create synthetic examples of the under-represented class in order to introduce diversity in the training set, using the following idea: Considering a vector from the under-represented class, we find its nearest neighbor, and compute the difference between these two vectors. We multiply this difference by a random number between 0 and 1, and add it back to the considered vector. This way, we generate a new data point along the line segment between two specific features. This technique effectively forces the decision region of the minority class to become more general.

This technique was quite efficient in this particular problem, and allowed us to gain on over-all performance, and to decrease the over-fitting problem.

3.1.3 Results

On Figure 1 we can see that even if our score was already acceptable without any over-sampling (around 0.82), adding over-sampling, and in particular SMOTE, allowed us to jump the gap between 0.82 and 0.86, which was crucial in terms of ranking. Here we specify "mean" score, because as explained in part 2, we run a single model on 20 different splits.

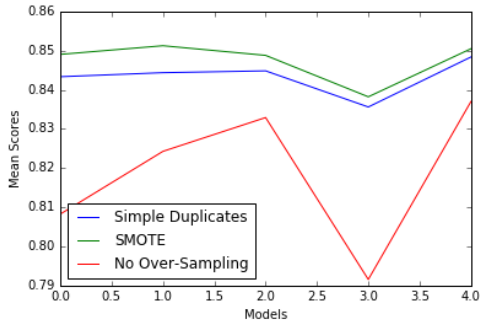

Figure 1: Mean Score for 3, 5, 7, 10 and 20 estimators on Adaboost

Figure 2 displays the variance of the 20 scores we get for one model. A bigger variance means that our model is less robust, as it performs unequally depending on the observations in the train set. Here we can see that over-sampling clearly helped training more robust model, without any major differences between simple duplicates and SMOTE.

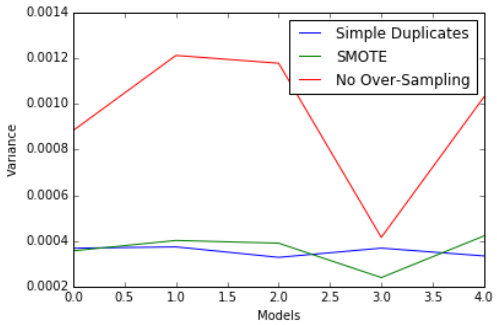

Figure 2: Variance for 3, 5, 7, 10 and 20 estimators on Adaboost

3.2 Feature engineering

3.2.1 Adding non linear operations for categorical features

The idea here is to use classical binary operations which are not linear between the different categorical features the datasets has. The rationale for this approach is that a combination of multiple categorical factors may result in pipe failure. Having no further information on the nature of the feature, we deemed interesting to carry out this idea. We implemented pairwise **and**, pairwise **or** and the same for triplets: X[c+u+w+'and'] = X[c]*X[u]*X[w] for example

Metrics	Raw data	Pair 'and'	Pair or'	Pair 'and' + 'or'	Triple 'and' + 'or'
Mean AUC Score	0.840	0.860	0.857	0.861	0.860
Variance AUC Score	4.38×10^{-4}	4.33×10^{-4}	7.95×10^{-4}	4.21×10^{-4}	$6.50 \times 10{-}4$

Table 3: Evolution of our estimated score with binary operations

3.2.2 Activations from neural network

The goal here was to use the last layers of the network as features. Using the network structure with one hidden layer we used the hidden layer activations as features for different classifiers. This gave slightly less good results (around 0.80) than with the non-linear operations approach. We tried deeper architectures and did not notice any major improvements. We decided to not pursue this lead.

4 Algorithms

4.1 Logistic Regression

The first stop for a classification task is a Logistic Regression. We thus attempted this method with different regularizations. Unsurprisingly, for a regularization that is too strong, the logistic regression does not perform very well, because the parameters are not fitted to the task (strong regularization is a small value for C).

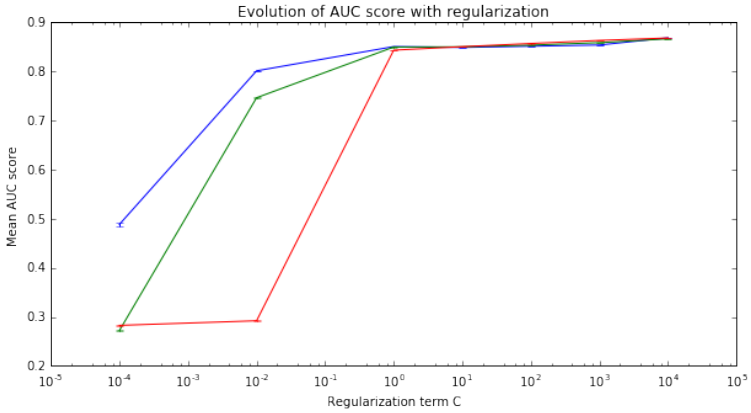

Figure 3: Evolution of performance of logistic regression for different regularization coefficients: in red the curve for raw features, in green for pairwise feature engineering, in blue for pairwise and triplets feature engineering

We notice how the feature engineering does not make much difference when setting a very low regularization (high value for C). High C means almost no regularization, which increases the likelihood of over-fitting. Indeed, we submitted our prediction for a classifier trained for maximum score given the curve on Figure 3, and performed a score of **0.8555** on the website's dataset. We predicted a score above 0.86, which may indicate some over-fitting from our part.

4.2 Trees

Decision Trees were a natural choice in this particular problem as most of our features were booleans, but also because they are known to be efficient with imbalanced datasets.
Classic decision trees and Random Forests fell short of our expectancies, as shown be figure 3, but boosting algorithm turned out to be quite successful.

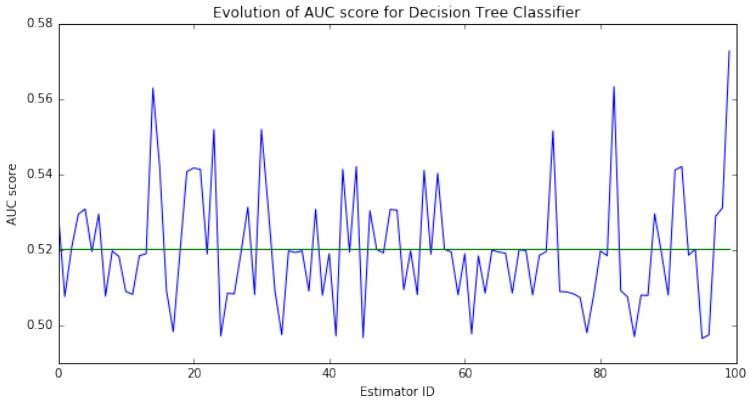

Figure 4: Decision Tree performance: in blue the score for the 100 Decision Trees we trained, in green the mean AUC score over the 100 trainings

4.2.1 AdaBoost

This technique was introduced as the first boosting technique by paper [2]. The core idea of AdaBoost is to train multiple classifiers (in our case decisions trees that are grown sequentially): each tree is grown using informations from previously grown trees, and is fit on a modified version of the original dataset. Those modifications consist of giving a greater weight to observations that were misclassified. Eventually, the final classifier is obtained thanks to a weighted average of all trees. This technique allows to slowly improve a decision tree classifier in areas where it does not perform well.

The algorithm that yielded our best performances for this challenge was an Adaboost algorithm, with a relatively low number of boosting stages (only 5).

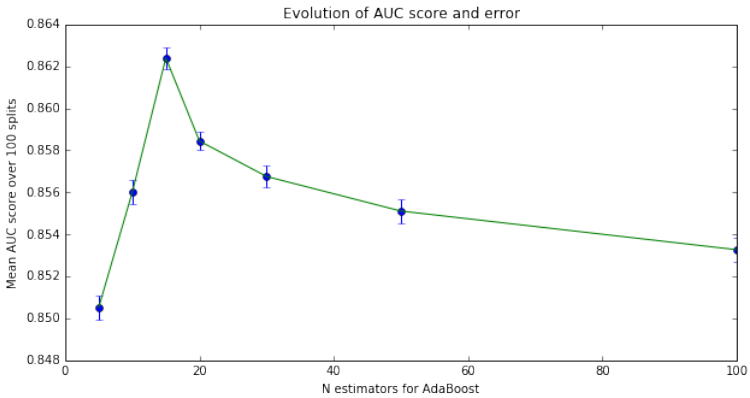

Figure 5: Evolution of Mean AUC score over 100 splits

4.2.2 Gradient Boosting

The great performances of Adaboost pushed us to try other decision tree boosting techniques. Gradient Boosting and Adaboost have many similarities. Here is an intuitive explanation to understand the difference: they both learn from previous models' errors and eventually yield a weighted sum of all models fitted during the process. The main difference lies in the fact that in Adaboost, the shortcomings of past models are identified via height-weight data points, whereas in Gradient Descent, they are identified by the gradient of the squared error loss function formed by the residuals. If this difference holds in an intuitive point of view, it doesn't in theory: the fact of re-weighting

some data points is equivalent to apply Gradient Boosting with an exponential loss function, which makes of Adaboost a refinement of Gradient Boosting.

In this project, we realized that the scikit learn library made some internal tuning when they developed their Adaboost method, which explains why in our special case, it worked better than their Gradient Boosting method with and exponential loss function.

4.2.3 Results

The Adaboost algorithm gave us our two most successful submissions, the first with a score of **0.878961**, with 10 estimators, using basic features and 'and' features, augmented with the SMOTE algorithm to a balance of fifty.
A second even better submission was made, with a score of **0.8845**, using the same parameters, and same data augmentation, but with basic features, 'and' features, and 'or' features.

4.3 Neural Networks

We investigated the neural network approach. Previous work [6] suggested using a network with one hidden layer. We built upon this architecture adding hidden layers and changing the number of nodes for each layer.

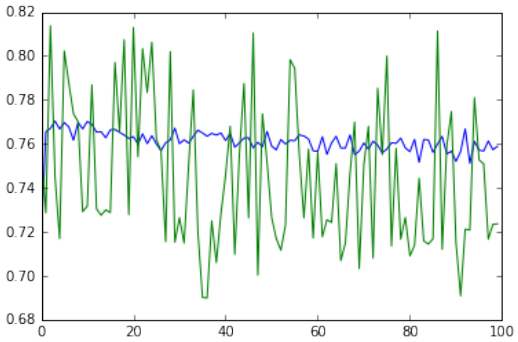

Figure 6: Evolution of accuracy over epochs; in blue accuracy on the training set, in green over the test set.

Figure 6 shows the instability of the network over the test set. Despite our iterations on the architecture and on hyper-parameters the neural network approach did not provide better performance compared to the other techniques. Also knowing our data did not exhibit any specific invariance or any multi-scale properties, we set this approach aside.

Conclusion

The main challenge of this task was to work with an imbalanced dataset. This problem was overcome thanks to our work on the score, which allowed us to have a more accurate idea if our algorithms' performances, and which led to scores around .80, without any over-sampling and very basic feature engineering. The steps after that were more incremental: working on data representation (generating non linear features) and on which algorithm makes the most sense, and working on over-sampling. Our final algorithm is thus a **AdaBoost Classifier with Decision Tree Classifier as base estimator** with the basic features and binary operations **and** and **or** for pairs and triplets. This approach yields a score of **0.8845**, which places us in second position in the challenge as of February 27th.

References

[1] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research 16, 321357*, 2002.

[2] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences , 55(1):119-139*, August 1997.

[3] Robert E. Schapire. *Explaining Adaboost*

[4] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, *An Introduction to Statistical Learning*, 2013.

[5] Jerome H. Friedman, Greedy Function Approximation: A Gradient Boosting Machine. *IMS 1999 Reitz Lectures*. 1999

[6] Moselhi, Osama and Shehab-Eldeen, Tariq, Classification of defects in sewer pipes using neural networks, *Journal of infrastructure systems*, 2000