All rights reserved © 2020 Luis Henrique PINTO

DETECTION OF FAILURES IN AIR PRESSURE SYSTEM (APS) BY USING DECISION TREE CLASSIFIERS

Luis H. PINTO — e-mail: luishenriquepinto.73@gmail.com

March, 2020

Abstract

This article describes the use of **Decision Tree Classifiers (DTC)** to predict the status of **Air Pressure Systems (APS)** installed on Scania heavy duty trucks, used to generate air pressurized to some vital systems like brakes and gear-changers. The predictions are based on data-set distributed by Scania under the terms of the **GNU — General Public License**. The main objective is to show the ability and versatility of DTC for dealing with a huge number of data-set attributes.

1. Introduction

Machine learning techniques are promoting a revolution in patter-recognition. Countless procedures allow the users to better understand processes based on data-sets and powerful computers can manage increasingly complex frameworks.

This article explains the use of **Decision Tree Classifiers (DTC)** to predict the status of **Air Pressures Systems (APS)** installed on Scania heavy duty trucks based on a set of 170 measurements recorded by sensors. The complexity of the problem depends on the adaptability of the DTC model to fit the data-set, the huge amount of clusters and the imbalance in the number of instances of each class representing the truck's status. These questions are described in the Secs. 2, 3 and 4 to make clear some well-established techniques used to compensate for defective data-sets.

Figure 1. Scania Heavy Duty Truck equipped with a Air Pressure System to generation of pressurized air

The chronology of the article offers a complete vision of the data-set and its characteristics, as well as the techniques used for fixing error associated to lack of data (Sec. 2) and re-sampling the complete set in order to contour the problem of imbalance (Sec. 3). A short-description of the DTC technique is also presented (Sec. 4) and finally the use of deep learning and **Python™** algorithms to solve the equation.

2. Data-Set Description

The data-set used in the current analysis contains 60,000 lines (or instances) of data directly collected from heavy trucks in everyday usage. The system in focus is the **Air Pressure System**

(APS) which generates pressurised air that is utilized in various functions in the trucks, such as braking and gear changes.

Each instance is composed by 170 attributes corresponding to physical measurements collected by sensors, and one extra-attribute labeled **class** associated to a specialized diagnostic of the truck attesting in case of a **'neg'**-value that there is a failure in some specific component of the APS, or a **'pos'**-value attesting a failure in the truck, but not in the APS. A short-view of the entire data-set is depicted in the Tab. 1.

	class	aa_000	ab_000	⋯	ee_009	ef_000	eg_000
0	'neg'	1.785e-03	0.003496	⋯	0.000000	0.000000	0.0
1	'neg'	7.696e-04	0.003496	⋯	0.000328	0.000000	0.0
2	'neg'	9.555e-04	0.003496	⋯	0.000112	0.000000	0.0
3	'neg'	2.793e-07	0.000000	⋯	0.000000	0.008299	32.0
4	'neg'	1.417e-03	0.003496	⋯	0.000266	0.000000	0.0
⋮	⋮	⋮	⋮		⋮	⋮	⋮
59995	'neg'	3.562e-03	0.003496	⋯	0.006255	0.000000	0.0
59996	'neg'	5.322e-05	0.003496	⋯	0.000000	0.000000	0.0
59997	'neg'	2.607e-06	0.000000	⋯	0.000000	0.000000	0.0
59998	'neg'	1.869e-03	0.003496	⋯	0.084986	0.000000	0.0
59999	'neg'	9.364e-04	0.003496	⋯	0.000035	0.000000	0.0

Table 1. Simplified view of the data-set format with classes and attributes.

Due proprietary reasons, the attribute names are modified. Sequences of two characters and three numerals from aa_000 to eg_000 are used in substitution to the originals.

A previous analysis of the data-set turns evident that there are two relevant aspects to be treated during the analysis:

- the imbalance between the number of instances corresponding to the **'neg'** (59,000 instances) and the **'pos'** (1,000 instances) classes; and
- 90% of the instances have **NaN**-values[1].

Both aspects impose consequences to the precision of the patter-recognition procedure, since the imbalance of the data-set tends to generate inaccurate responses when processing information from the class with the smallest sample; and the **NaN**-values are obviously a source or incertitude by themselves.

In order to minimize the effects of the **NaN**-values, a simple substitution technique is used, in which the **NaN** associated to an attribute are replaced by the average of the valid values. Certainly it is not a perfect technique, but in a statistical context involving a huge amount of data, the error associated to the choice is neglected.

The approach to solve the problem related to the imbalance of the data-set is shown in the Sec. 3.

Another question related to the data-set is the not so evident correlation between the the attributes, and a complex mix of the classes **'neg'** and **'pos'** without forming isolated clusters, as shown in the Fig. 2.

The complete set of pair-plots with the correlation between the 170 attributes is not shown in this document.

[1] The not-a-number (or **NaN**) code represents lack of information due some problem during the data-record.

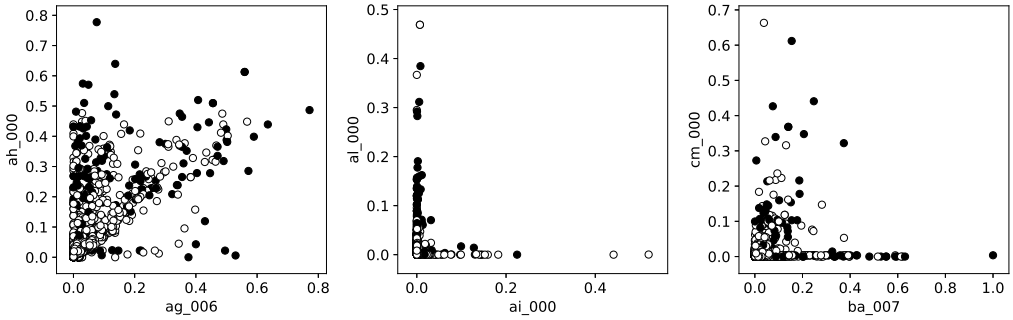

Figure 2. Aleatory set o pair-plots showing the correlations between three pairs of attributes from the data-set and the multitude of small-isolated clusters — white circles are related to the **'neg'** class, and black circles to the **'pos'** class.

3. Re-sampling Imbalanced Data-Sets

One of the major issue when dealing with imbalanced data-sets relates to the metrics used to evaluate the model. Using simpler metrics like the **accuracy score** can be misleading. In a data-set with highly imbalanced classes, if the model always predicts the most common class without performing any analysis of the features, it will still have a high accuracy rate, but obviously illusory.

A widely adopted technique for dealing with highly imbalanced data-sets is called **re-sampling**. It consists of removing samples from the majority class (called **under-sampling**) or adding more examples from the minority class (called **over-sampling**).

The choice to correct the imbalance problem in this article is the use of the **Synthetic Minority Oversampling Technique (SMOTE)** technique. SMOTE consists of synthesizing elements for the minority class, based on those that already exist. It works randomly picking a point from the minority class and computing the **k-nearest** neighbors for this point, where **k** is an integer parameter chosen by the user. The synthetic points are added between the chosen point and its neighbors.

This technique also have its weaknesses. The simplest implementation of over-sampling is to multiply random records from the minority class, which can cause overfitting and consumption of resources for processing data. In the current case, the number of instances reaches 118,000.

4. The Decision Tree Learning (DTL)

Decision Tree Learning (DTL) is a method commonly used in data mining where the goal is to create a model that predicts the value of a target variable based on several input variables, resulting in a **Decision Tree Classifier (DTC)** or a **Decision Tree Regressor (DTR)** depending on the predicted outcome:

- **Decision Tree Classifier** — when the predicted outcome is a class (discrete) to which the data belongs; and
- **Decision Tree Regressor** — when the predicted outcome can be considered a real number.

The architecture of a decision tree is build by using four types of structures: the **root-node (root)** corresponding to the input variables space; the **rule-nodes (non-leafs)** associated to the decision-making process; the **internal-nodes** used to store the input feature; and the **terminal-nodes (leafs)** corresponding to the target variable space (or classes). The internal and terminal-nodes are assigned with the input features and the classes, respectively. Subsequently, the internal-nodes are linked to the rule-nodes in order to form the decision-making process.

The tree is built by splitting the source set, constituting the root-node of the tree, into subsets — which constitute the **successor children**. The splitting is based on a set of splitting rules operated by the rule-nodes and based on classification features. This process is repeated on each derived subset in a recursive manner called **recursive partitioning**. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees is an example of a **greedy algorithm**, and it is by far the most common strategy for learning decision trees from data. The Fig. 3 shows a simplified version of a DTC with four input features and three outcome classes, according to a four-level depth format.

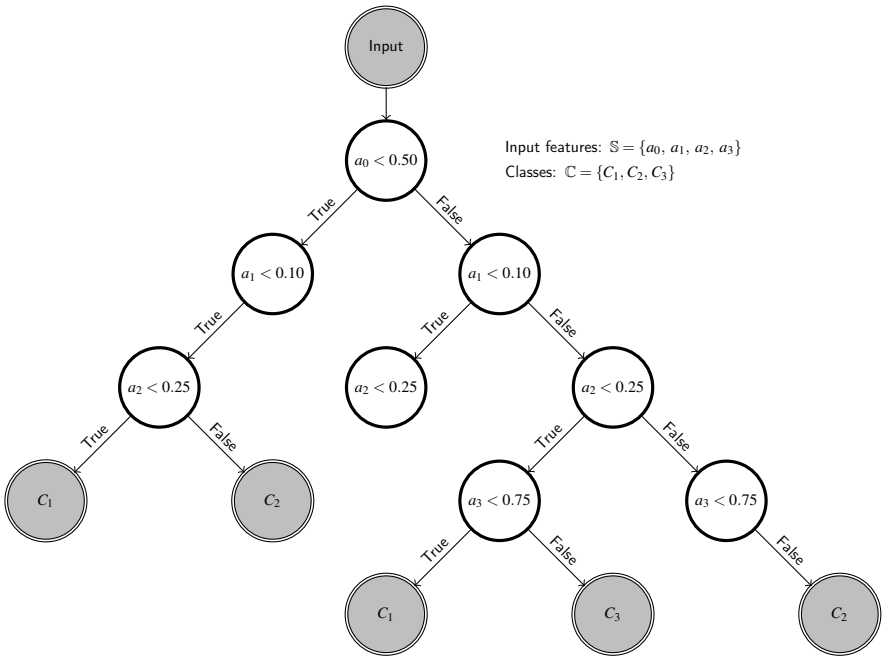

Figure 3. Simplified diagram of a Decision Tree Classifier (DTC) with 4-level depth.

As mentioned in the first section of this article, the main objective of the current development is to classify the status of the APS in the classes **'pos'** and **'neg'** as a function of the 170 input features collected for each truck, so according to the previous definition, the solution is based on a **classification tree analysis** method.

5. Simulation

The test routines for the solution of the problem described in the Sec. 1 are developed in a **Python™** environment using the **Scikit-Learn — Machine Learning in Python** libraries and following the steps depicted in the Fig. 4.

The steps 2 and 3 are extensively commented in the Secs. 2 and 3. They have the sole objective of fixing the data-set with respect to the **NaN**-values and the imbalance between the classes **'pos'** and **'neg'**. The functions chosen to perform this task was `SimpleImputer` which substitute the **NaN**-values present in the data-set by the **average** of the correlated attribute-values; and `RandomOverSampler` which over-sample the minority class (i.e.: the **'neg'** class) by creating intermediary points between two nearest data according to a **SMOTE — Synthetic Minority Over-Sampling Technique**.

The fourth step is the central procedure related to the solution of the problem. The function `RandomForestClassifier` is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the data-set and uses averaging to improve the predictive accuracy and

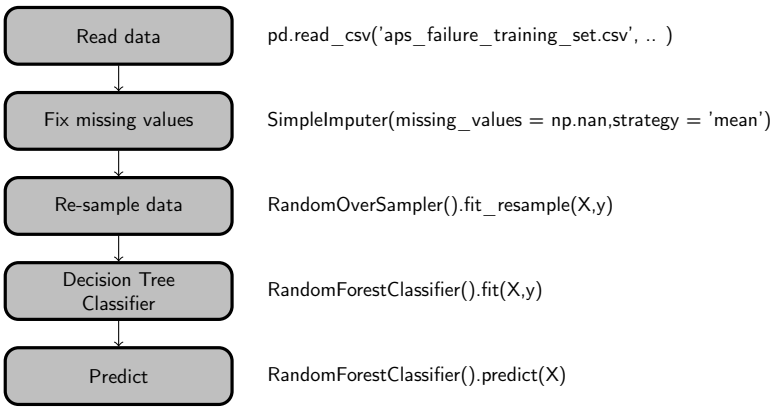

Figure 4. Caption

control over-fitting. The `fit` function is the responsible for searching the solution that better fits the 170 attributes and the 60,000 instances presented in the data-set with respect to the **accuracy** parameter. The results of the simulation including the artificially re-sampled values for the class **'neg'** are shown in the Tab. 2. During the simulation 70% of the instances in the data-set are presented to the system to perform the calibration procedure, and the remaining 30% are used as a sample to test the accuracy of the proposed solution.

	precision	recall	f1-score	support
'pos'	1.00	1.00	1.00	17,700
'neg'	1.00	1.00	1.00	17,700
accuracy			1.00	35,400
macro avg	1.00	1.00	1.00	35,400
weighted avg	1.00	1.00	1.00	35,400

Table 2. Classification report comparing true-values with predicted-values for the classes **'pos'** and **'neg'** by using **Decision Tree Classifiers** in a **Random Forest** configuration.

The final accuracy of the model attested by an extra data-set including real samples with 16,000 instances is shown in the Tab. 3. This data-set also present **NaN**-values and imbalance between classes, but only the first issue is corrected once the objective is testing the performance of the model under real situation (i.e.: with a minimum of artificially-made data).

	precision	recall	f1-score	support
'pos'	1.00	1.00	1.00	15,625
'neg'	0.86	0.85	0.86	375
accuracy			0.99	16,000
macro avg	0.93	0.92	0.93	16,000
weighted avg	0.99	0.99	0.99	16,000

Table 3. Classification report comparing true-values with predicted-values for the classes **'pos'** and **'neg'** by using **Decision Tree Classifiers** in a **Random Forest** configuration.

On both Tabs. 2 and 3 can be noted that the accuracy is close to 99%. This is a good parameter for well-balanced data-sets, but despite the accuracy and the fact that the model

performs with high precision in classifying trucks with status **'pos'** (i.e.: trucks presenting failures in the APS), in real situations, it fails for 17% of the trucks with status **'neg'** as can be noted through the **precision** parameter (i.e.: trucks presenting failures, but not in the APS). This fact does not detract the FTC with respect to its performance. For balanced data-sets (see Tab. 2) the accuracy and the precision are completely aligned to the expectations.

Despite the lack of precision for predicting **'neg'** classes, the use of DTC for the classification of trucks with problems in the APS is extremely satisfactory, what makes DTC an exceptional tool to guide repair tasks in a dealership workshop for example, or even to be integrated in on-board fault alert systems, saving labor and giving early diagnostics.

> A complete version of the **Python™** files used in the simulations can be found in the **GitHub** link APS Failure at Scania Truck Data-Set, including a detailed description depicted in the README.md file.

6. About the Author

Luis H. Pinto is Mechanical Engineer, specialized in Automotive Engineering, Business Management, and Thermal Sciences, 21 years experienced as technical manager with emphasis on optimization of manufacturing processes of automotive industries, researching and development of methodological methods for qualitative and quantitative analysis of equipment and process reliability, including the use of deep learning concepts.

luishenriquepinto.73@gmail.com

References

[1] François Chollet et al. *Keras: The Python Deep Learning Library*. 2015.

[2] Simon Haykin. "Feed-forward Neural Networks: An Introduction". In: (2004), pp. 1–16.

[3] Tony Lindgren and Jonas Biteus. *APS Failure at Scania Trucks Data Set*. 2016.

[4] Luis Pinto. "Data-driven Method for the Prediction of Equipment Remaining Useful Life — Clustering and Neural Network Pattern-Recognition". In: (2019).

[5] WikiPedia. *Decision Tree Learning*. 2020.