Technical Disclosure Commons

Defensive Publications Series

October 01, 2018

MACHINE LEARNING FOR THING DETECTION, BEHAVIOR ANALYTICS, AND THREAT DETECTION: AN EFFICIENT ELIMINATION METHOD OF REDUNDANT FEATURE-SUBSETS

Tomas Komarek

Jan Brabec

Lukas Machlica

MACHINE LEARNING FOR THING DETECTION, BEHAVIOR ANALYTICS, AND THREAT DETECTION: AN EFFICIENT ELIMINATION METHOD OF REDUNDANT FEATURE-SUBSETS

AUTHORS:
Tomas Komarek
Jan Brabec
Lukas Machlica

ABSTRACT

Techniques are described herein to efficiently detect redundant features in a machine learning process. The techniques are able to compute feature redundancy not only for a single feature at a time, but for any subset of features without the need to naively train and evaluate a classifier for each combination of features.

DETAILED DESCRIPTION

In classification systems, often a large amount of features is engineered by domain experts. The features are introduced to a learning algorithm, which may during the training process neglect some of the features while promote others. However, in most of the cases the classifiers, acting as a black box, do not directly indicate importance of a feature. Therefore, even if a feature has almost no impact on a classification performance, it still has to be extracted. In network intrusion detection systems, feature extraction related to external intelligence, e.g. querying external database or crawling a website and parsing its content, may become quite expensive, both computationally and financially. Paying for several blacklist may not be needed if most of the usable content is available in bot external database and websites.

To tackle this problem, methods such as permutation feature importance or feature selection algorithms are proposed. The main disadvantage of these methods is that they fail whenever multiple features are correlated, e.g. using more than two blacklist with similar content, and/or they are not able to easily incorporate different types of features at the same time, e.g. categorical, discrete and continuous.

Additional downsides of redundant features are (1) they slow down model training and testing, (2) they make it harder to interpret and explain model’s decisions, and (3) they deplete memory resources, e.g., the need to store training/testing datasets.

One of the methods to remove redundant features is based on feature selection. In a filter method, a Conditional Mutual Information Maximization (CMIM) algorithm may be employed. However, the CMIM algorithm can be applied only for binary target variables and binary or discrete features. The CMIM algorithm only uses pair-wise feature independence.

In another embedded method, a random forest algorithm based on variable or permutation importance may be used. Whenever two features are correlated (even identical), none of them is identified as redundant by this approach.

In a wrapper method, a naïve approach may be adopted, which eliminates features one by one. To eliminate the first feature (e.g. out of 40 features in total), it requires training and testing 40 models on various feature subsets each time with one missing feature, and comparing their performances with a model trained and tested on the full-feature set. A feature missing in a feature subset that produces a model with the closest performance to the original full-feature model then corresponds to the most irrelevant feature. These steps can be repeated to eliminate other features. The one-by-one elimination is computationally very intensive as it requires to train and test lots of models. Additionally, this approach explores only pair-wise dependences. To explore a higher order dependences (e.g. triplets of features), it would require training and testing exponentially more models.

Techniques disclosed herein provide an efficient method, which is able to compute feature redundancy not only for a single feature at a time, but for any subset of features without the need to naively train and evaluate a classifier for each combination of features.

One technique is to use the power of random feature selection in random forests. More precisely, a random forest with increased number of trees is trained, where each tree is built only on a subset of features, e.g. half of all the available features. For example, an application is to assess redundancy of three features, namely {A, B, C}. To accomplish this, only those trees are selected from the forest that do not use A, B nor C and form a new forest RF2. Further, a forest RF1 is formed from trees that contain each of the three features. If each tree contains half of the features picked randomly, and number of trees in the forest is T = 96, then both forest will approximately have $(1/2)^3*96 = 12$ trees. These two forests score the testing samples from D_test, and the difference between performance of RF1 and RF2 is computed. If the absolute difference is arbitrarily small or lower than a given accepted performance decrease, then {A, B, C} are redundant. Note that the scoring of D_test has not to be repeated for each comparison; it may be performed only once for all the trees in the original forest. Decisions of the individual trees are remembered for individual samples. The trees and the original dataset are not needed when redundancy of any subset of the original feature set F is addressed.

The proposed method for eliminating feature subsets can be summarized as follows:

INPUT:

E – a number of eliminated features.

D_train, D_test – training and testing dataset with F features and M testing samples.

T – a number of decision trees that are usually trained within a random forest to obtain a good prediction performance. For example, it is sufficient that T is set at 20 to 100.

OUTPUT:

O - an ordered list of feature subsets of size E. The lower value is assigned to a feature subset, the more redundant/irrelevant the subset is.

PROCEDURE:

1. Randomly generate $N = T * 2^E$ feature subsets of size F/2 (without feature replacement).

2. On each generated feature subset, train a decision tree using D_train.

3. Compute matrix of predictions P of size (M, N), where an element (m, n) corresponds to a prediction of n-th tree on m-th testing sample from D_test.

4. For each possible subset S of E features (out of F feature in total) do:

    a. Compose two random forests.

        i. RF1: Identify decision trees (columns of matrix P) that were trained on feature subsets containing all features from S.

        ii. RF2: Identify decision trees (columns of matrix P) that were trained on feature subsets that do not contain any feature from S.

    b. Using matrix P aggregate decisions of both random forests over all testing samples (using majority/soft-voting).

c. Evaluate performance of both random forests with respect to true labels (e.g. Area under Precision-Recall curve).

d. Push S into O with a value that equals to absolute value of a difference between the two evaluation scores.

5. Sort O according to the assigned values in ascending order.

These steps are illustrated in Figure 1 below.

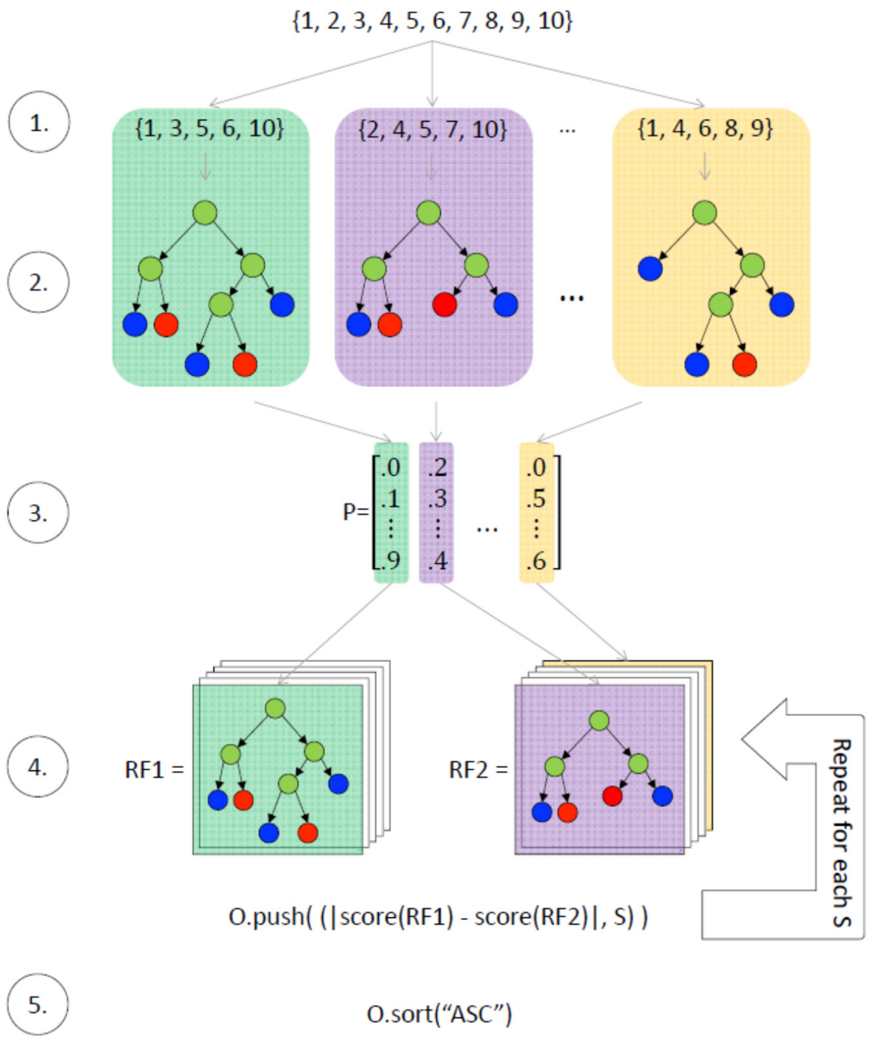

Figure 1

The exact formula for probability that a single tree contains a subset of k features is:

$$\prod_{i=0}^{k-1} \frac{\frac{F}{2} - i}{F - i}$$

When F is sufficiently large compared to i, it can be reasonably approximated by a simpler formula $(1/2)^k$. This approximation is used in the text above.

A small difference in performance between random forests that use S and that do not use S indicates that the features in S are irrelevant or correlated with other involved features. Optionally, irrelevant features can be eliminated using Variable Importance method and the proposed method can be applied to identify only the redundant features. The proposed method is very efficient as compared to the naive approach since the most computationally heavy part, steps 2 and 3 in the procedure, which are model training and testing on D_train, D_test, is performed only once. Once the matrix of predictions P is computed, the method can be run again for different sizes of S from 1 to E without the need to recompute the matrix P. All types of features (continuous, discrete, categorical, etc.) are handled naturally. Scoring is performed from the perspective of a model (random forest) that can also be used in the final deployment. In Addition, a standard implementation of random forests can be used. The techniques are easily extendable to multi-class problems.

In summary, techniques described herein provide an efficient elimination scheme for measuring the redundancy of features using tree-based machine learning models. The main advantage over existing methods for feature selection or feature importance is that the illustrated method is able to compute feature redundancy not only for a single feature at a time, but also for any subset of features without the need to naively train and evaluate a classifier for each combination of features.