

Spis treści:

- [1. Instalacja](#)
- [2. Opis](#)
- [3. Przetwarzanie danych](#)
- [4. Modele](#)
- [5. Raport](#)

## 1. Instalacja

---

```
devtools::install_github("chmjelek/scania")
```

Przy zapytaniu:

```
Enter one or more numbers, or an empty line to skip updates:
```

Należy wpisać **3** i potwierdzić klawiszek **enter**.

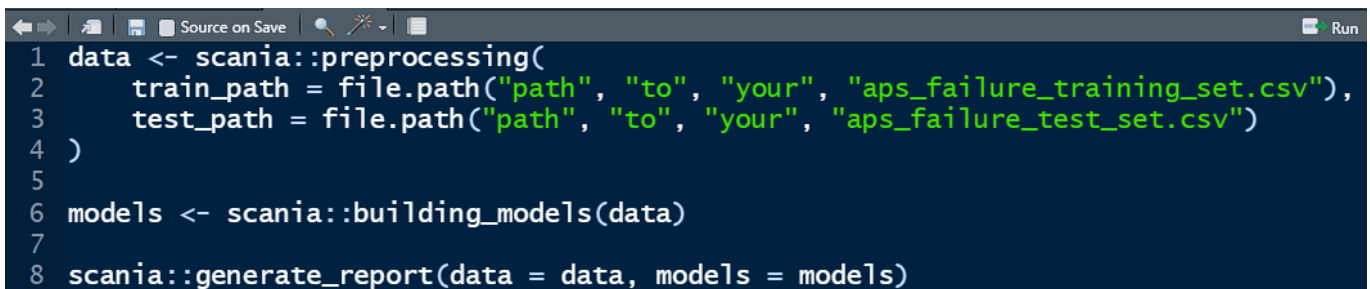
Po instalacji stwórz skrypt i wywołaj pakiet używając **scania::** albo **library(scania)**

Aby zobaczyć pełną listę eksportowanych funkcji:

```
library(scania)
ls("package:scania")
```

Dane należy pobrać: [link](#).

Przykładowe wywołanie:



```
1 data <- scania::preprocessing(
2   train_path = file.path("path", "to", "your", "aps_failure_training_set.csv"),
3   test_path = file.path("path", "to", "your", "aps_failure_test_set.csv")
4 )
5
6 models <- scania::building_models(data)
7
8 scania::generate_report(data = data, models = models)
```

## 2. Opis

---

Projekt dotyczy zastosowania technik uczenia maszynowego w przemyśle produkcyjnym i samochodowym. Projekt opiera się na danych uzyskanych z czujników samochodów ciężarowych Scania w celu przewidywania awarii związanych z układami ciśnienia powietrza.

Dane z czujników zostały dostarczone przez ekspertów i nie podali oni żadnych nazw atrybutów z powodów zastrzeżonych. Tak więc, otrzymaliśmy dane bez nazw atrybutów z repozytorium uczenia maszynowego [UCI](#). Dane mają 171 atrybutów, wiele brakujących wartości. Istnieje również macierz kosztów błędnej klasyfikacji, a celem jest zredukowanie tych kosztów.

Głównym celem jest sklasyfikowanie awarii w samochodach ciężarowych Scania, czy są one spowodowane przez podzespoły związane z APS (Air Pressure System) czy przez inne podzespoły, które nie są związane z APS. Dane mają tylko dwie klasy, tzn. pozytywną lub negatywną. Jest to więc problem klasyfikacji binarnej.

Z problemem tym związany jest również koszt błędnej klasyfikacji:

$\text{\$koszt}_{\{1\}}$ ; wykonania niepotrzebnej kontroli (false negative), wynosi 10

$\text{\$koszt}_{\{2\}}$ ; niewykrycia wadliwej ciężarówki (false positive), co może spowodować awarię, wynosi 500

Stąd koszt ogólny wynosi:

$$\text{\$koszt} = \text{\$koszt}_{\{1\}} \cdot n_{\{1\}} + \text{\$koszt}_{\{2\}} \cdot n_{\{2\}}$$

gdzie:

- $n_{\{1\}}$  - liczba niepotrzebnych kontroli
- $n_{\{2\}}$  - liczba niewykrytych wadliwych ciężarówek

Naszym zadaniem jest minimalizacja tego kosztu.

Dane są bardzo nie zrównoważone, aby poradzić sobie z tym problemem, zastosowano różne metody samplingu. Poniżej znajduje się krótki opis naszego podejścia.

## 3. Przetwarzanie danych

---

### Dane

Zbiór treningowy zawierał w sumie 60000 przykładów, z czego 59000 należy do klasy negatywnej, a 1000 do pozytywnej.

Klasa była objaśniana przez 170 atrybutów.

Pakiet pozwala na wstępne przetworzenie danych, atrybuty z:

- więcej niż  $x\%$  zer
- więcej niż  $y\%$  brakujących danych
- o odchyleniu standardowym poniżej  $z$

mogą zostać usunięte.

Pozostałe brakujące dane atrybutu uzupełnić jego medianą.

## Sampling

Zdecydowaliśmy się użyć upsamplingu dla klasy mniejszej (pozytywnej, z 1000 przykładów), oraz undersamplingu dla klasy większej (negatywnej, z 59000 przykładami), tak aby liczba przykładów w każdej klasie była równa.

Upsampling został wykonany przez kopiowanie oryginalnych klas pozytywnych, undersampling przez losowe wybieranie przykładów z klasy negatywnej, aby dopasować liczbę przykładów w klasie pozytywnej.

Pakiet pozwalała na tworzenie dowolnej ilości kopii klasy mniejszej (wraz z mniejszeniem klasy większej).

## Szum

Do skopiowanych przykładów klas pozytywnych dodawany jest szum (wartość liczbowa pochodząca z rozkładu normalnego danej cechy). Zabieg ten ma na celu zróżnicowanie skopiowanych przykładów klas pozytywnych. Chcieliśmy sprawdzić, jaki wpływ na model będzie miał taki proces.

Proces dodawania szumu:

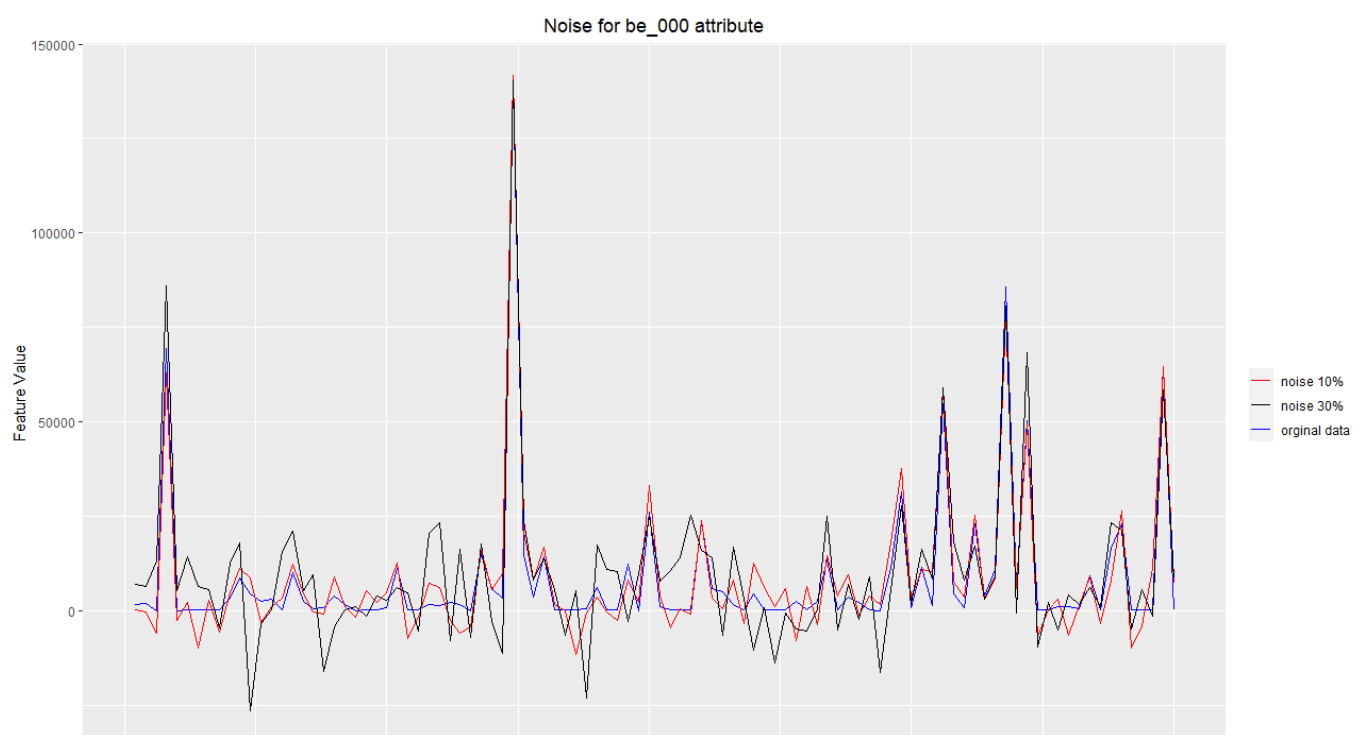
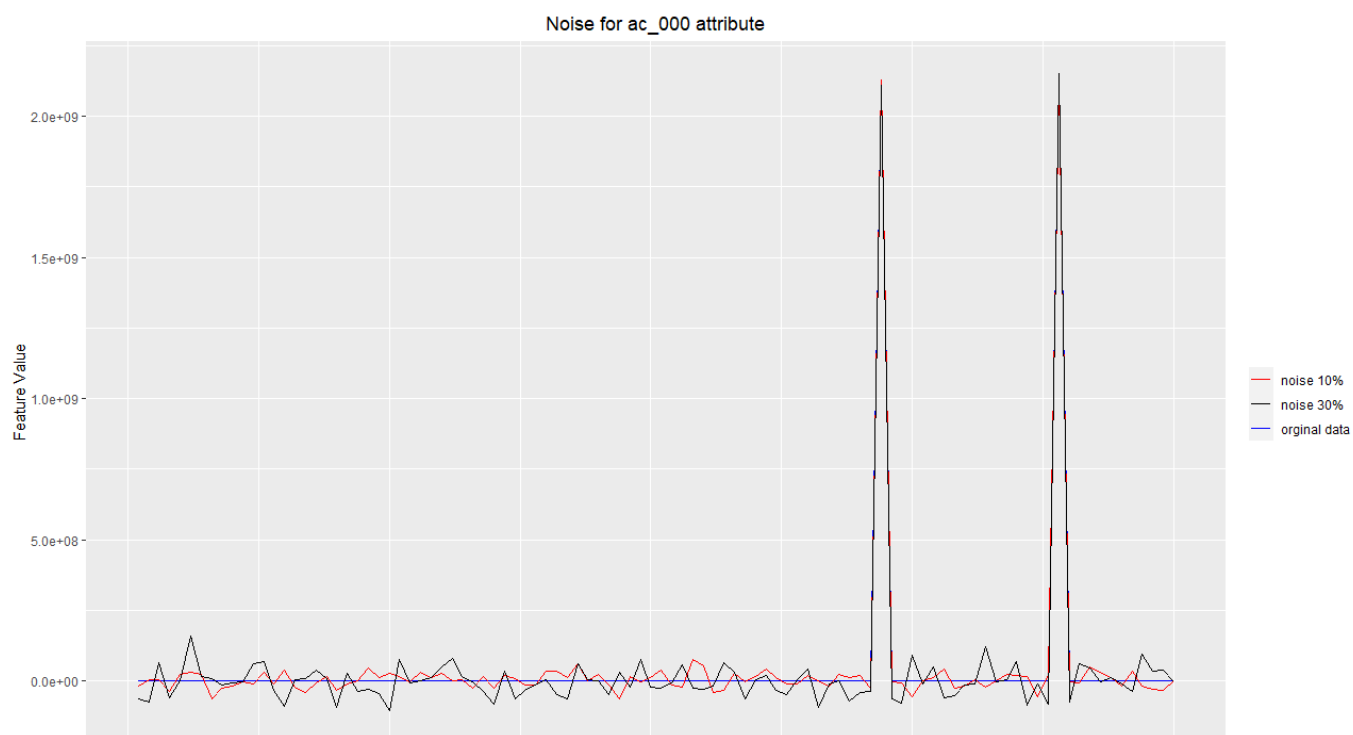
- ustal współczynnik skali  $p$
- znajdź odchylenie standardowe  $\text{std}$  każdego atrybutu  $a$
- dla każdej instancji  $x$  atrybutu  $a$  oblicz  $\text{szum}$  według podanego wzoru:
  - $\text{noise} = N(0, \text{std}(x) * p)$ .
- dodaj  $\text{noise}$  do każdej instancji  $x$

Współczynnik skali  $p$  określa stopień szumu, który może być dodany do danych. Im wyższa wartość  $p$ , tym większy szum jest dodawany.

Dla każdego egzemplarza oryginalnej klasy pozytywnej można było wybrać inny poziom szumu.

Kilka przykładów dodawania szumu do atrybutu:





W naszym pakiecie powyższe czynności wykonuje funkcja `preprocessing`

```
preprocessing {scania}
```

## Prepares data for the models

### Description

Prepares data for the models

### Usage

```
preprocessing(train_path, test_path, noise_level = c(0, 0.1, 0.2), ...)
```

### Arguments

**train\_path** chr; path to training dataset  
**test\_path** chr; path to testing dataset  
**noise\_level** float or vector of floats; by default c(0, 0.1, 0.2)  
**zeros\_perc** float; zeros ratio in attribute (above that value attribute will be removed); by default 0.95  
**na\_perc** float; NA ratio in attribute (above that value attribute will be removed); by default 0.95  
**sd\_val** float; sd value of attribute (below that value attribute will be removed); by default 1

### Value

list of 4; preprocessed, noises, train, test

- preprocessed - preprocessed data.frames
- noises - list of dataframes with noised records
- train - training data table
- test - test data frame

## 4. Modele

Wytrenowaliśmy dane za pomocą 5 różnych zaawansowanych technik uczenia maszynowego, takich jak **SVM** z jądrem liniowym, radialnym i wielomianowym, **RandomForest**, algorytm **XGBoost** i obliczyliśmy koszt na podstawie macierzy konfuzji wygenerowanej przez modele. Spośród wszystkich algorytmów **RandomForest** zminimalizował koszt znacznie mniej niż pozostałe algorytmy.

Naszym celem jest obliczenie kosztów związanych z błędami złej klasyfikacji i znalezienie najlepszego modelu z minimalnym kosztem. Tak więc, aby obliczyć koszt, generujemy macierz konfuzji dla każdego modelu i porównujemy je na końcu.

name	Accuracy	Kappa	Recall	Precision	tn	tp	fn	fp	TotalCost
RandomForest	96.33	53.74	96.32	99.92	15050	363	575	12	11750
XGBoost	97.36	61.51	97.43	99.87	15223	355	402	20	14020
SVM_linear	96.84	56.64	96.92	99.84	15144	350	481	25	17310
SVM_radial	91.12	31.34	90.96	99.94	14213	366	1412	9	18620
SVM_polynomial	96.99	57.09	97.15	99.76	15179	339	446	36	22460

Do ewaluacji modeli wykorzystano 9 współczynników:

- Accuracy =  $(tn + tp / tn + tp + fn + fp)$  - ile razy poprawnie wskazaliśmy klasę - nie zawsze model z najwyższą dokładnością daje minimalny koszt - (false positive kosztuje 50 razy więcej niż false negative)
- Kappa - dokładność klasyfikacji, z tą różnicą, że jest ona znormalizowana na poziomie bazowym losowego przypadku na zbiorze danych
- Recall =  $(tp / tp + fn)$  - stosunek poprawnych pozytywnych przewidywań do całkowitej liczby pozytywnych przykładów
- Precision =  $(tp / tp + fp)$  - stosunek poprawnych przewidywań pozytywnych do całkowitej liczby przewidywań pozytywnych
- tn - poprawne wykrycia klasy negatywnej
- tp - poprawne wykrycia klasy pozytywnej
- fn - niewykrycie klasy negatywnej (koszt 10)
- fp - niewykrycie klasy pozytywnej (koszt 500)
- TotalCost - koszt całkowity

W naszym pakiecie powyższe czynności wykonuje funkcja `build_models`

```
building_models {scania}
```

## Builds models

### Description

Builds models

### Usage

```
building_models(
  data,
  gamma = 0.1,
  cost = 0.1,
  ntrees = 50,
  mtry = 5,
  importance = TRUE,
  ...
)
```

### Arguments

<code>data</code>	list of 3; noises, train, test
<code>gamma</code>	float; svm; parameter needed for all kernels except linear (default: $1/(\text{data dimension})$ )
<code>cost</code>	float; svm; cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation
<code>ntrees</code>	int; RandomForest; number of trees to grow
<code>mtry</code>	int; RandomForest; number of variables randomly sampled as candidates at each split
<code>importance</code>	bool; RandomForest; should importance of predictors be assessed
<code>eta</code>	float; xgboost; control the learning rate; scale the contribution of each tree by a factor of $0 < \eta < 1$ ; by default 0.3
<code>nrounds</code>	int; xgboost; number of iterations; lower value for eta implies larger value; by default 75
<code>max_depth</code>	int; xgboost; maximum depth of a tree; by default 6
<code>subsample</code>	float; xgboost; subsample ratio of the training instance; 0.5 prevents overfitting; by default 0.5
<code>colsample_bytree</code>	float; xgboost; subsample ratio of columns when constructing each tree; by default 0.5
<code>eval_metric</code>	chr; xgboost; evaluation metrics for validation data; by default "error"
<code>objective</code>	chr; xgboost; specify the learning task and the corresponding learning objective; by default "binary:logistic"

### Value

data.frame; containing models

## 5. Raport

---

Pakiet pozwala również na generowanie raportu.

Funkcja `generate_report`:

```
generate_report {scania}
```

## Generates html report

### Description

Generates html report

### Usage

```
generate_report(  
  data,  
  models,  
  to_plot = c("aa_000", "ba_003", "cn_005"),  
  output_dir = "~/scania/"  
)
```

### Arguments

<code>data</code>	list of 4; preprocessed, noises, train, test
<code>models</code>	data.frame; output models
<code>to_plot</code>	chr; a vector of attributes names' to plot
<code>output_dir</code>	chr; path to save the report