02.Uczenie modelu

Mamy już przygotowane w poprzenim kroku zadanie regresji:

```
autoMpgTask = readRDS('data/01_task.RDS')

print(autoMpgTask)
```

```
## Supervised task: auto_mpg
## Type: regr
## Target: mpg
## Observations: 398
## Features:
## numerics  factors  ordered
##        9        0        0
## Missings: TRUE
## Has weights: FALSE
## Has blocking: FALSE
```

czas na wytrenowanie pierwszych modeli.

Przydatne informacje:

- http://mlr-org.github.io/mlr-tutorial/devel/html/learner/index.html
- https://mlr-org.github.io/mlr-tutorial/release/html/integrated_learners/index.html
- http://mlr-org.github.io/mlr-tutorial/devel/html/tune/index.html
- http://mlr-org.github.io/mlr-tutorial/devel/html/parallelization/index.html

Jaki model wybrać?

Sprawdźmy najpierw jakie modele regresyjne (metoda `listLearners`):

class	name	short.name
regr.IBk	K-Nearest Neighbours	ibk
regr.cforest	Random Forest Based on Conditional Inference Trees	cforest
regr.ctree	Conditional Inference Trees	ctree
regr.cvglmnet	GLM with Lasso or Elasticnet Regularization (Cross Validated Lambda)	cvglmnet
regr.featureless	Featureless regression	featureless
regr.gausspr	Gaussian Processes	gausspr
regr.glm	Generalized Linear Regression	glm
regr.glmnet	GLM with Lasso or Elasticnet Regularization	glmnet
regr.km	Kriging	km
regr.ksvm	Support Vector Machines	ksvm
regr.lm	Simple Linear Regression	lm
regr.mob	Model-based Recursive Partitioning Yielding a Tree with Fitted Models Associated with each Terminal Node	mob
regr.nnet	Neural Network	nnet
regr.randomForest	Random Forest	rf
regr.randomForestSRC	Random Forest	rfsrc

class	name	short.name
regr.rpart	Decision Tree	rpart
regr.rvm	Relevance Vector Machine	rvm
regr.svm	Support Vector Machines (libsvm)	svm
regr.xgboost	eXtreme Gradient Boosting	xgboost

Sprawdźmy co jest dostępne dla naszego zadania:

class	name	short.name
regr.cforest	Random Forest Based on Conditional Inference Trees	cforest
regr.ctree	Conditional Inference Trees	ctree
regr.featureless	Featureless regression	featureless
regr.randomForestSRC	Random Forest	rfsrc
regr.rpart	Decision Tree	rpart

Trenowanie

Spróbujmy wytrenować proste drzewo losowe (`rpart`) stosując do tego walidację krzyżową:

```
## rmse.test.rmse   rmse.test.sd
##      3.5908830      0.5430745
```

Strojenie parametrów

Parametry naszego algorytmu to (ich wyjaśnienia `?rpart.control`):

```
getParamSet('regr.rpart')
```

```
##                     Type len  Def   Constr Req Tunable Trafo
## minsplit         integer   -   20 1 to Inf   -    TRUE     -
## minbucket        integer   -    - 1 to Inf   -    TRUE     -
## cp               numeric   - 0.01   0 to 1   -    TRUE     -
## maxcompete       integer   -    4 0 to Inf   -    TRUE     -
## maxsurrogate     integer   -    5 0 to Inf   -    TRUE     -
## usesurrogate    discrete   -    2    0,1,2   -    TRUE     -
## surrogatestyle  discrete   -    0      0,1   -    TRUE     -
## maxdepth         integer   -   30  1 to 30   -    TRUE     -
## xval             integer   -   10 0 to Inf   -   FALSE     -
```

Dodamy strojenie parametrów do procesu uczenia:

1. tworzymy `ParamSet` dla parametrów `cp`,
2. wybieramy strategię strojenia - np. po hipersiatce sprawdzając 20 różnych wartości,
3. "opakowujemy" nasz algorytm uczący dodając strojenie,
4. powtarzamy eksperyment dla zadania `autoMpgTask`.

```
## rmse.test.rmse   rmse.test.sd
##      3.3272988      0.4622236
```

Aby przyspieszyć obliczenia możemy zastosować zrównoleglenie przy pomocy pakieru `parallelMap`. Zmienna `level` kontroluje poziom, na którym obliczenia będą prowadzone równolegle - schemat jest następujący:

```
parallelMap::parallelStartMulticore(level = 'mlr.resample')
doTraining()
parallelMap::parallelStop()
```

Porównanie z modelem liniowym

Model liniowy nie wspiera brakujących danych - można je uzupełniać w trakcie uczenia: http://mlr-org.github.io/mlr-tutorial/devel/html/impute/index.html

```
## rmse.test.rmse   rmse.test.sd
##      3.3967104      0.4335909
```

Porównanie kilku modeli

Z naszych modeli możemy utworzyć benchmark:

http://mlr-org.github.io/mlr-tutorial/devel/html/benchmark_experiments/index.html

```
##    task.id       learner.id rmse.test.rmse rmse.test.sd
## 1 auto_mpg  regr.lm.imputed       3.336596    0.3976971
## 2 auto_mpg regr.rpart.tuned       3.230362    0.5304323
```

Dodatkowe ćwiczenia:

1. jakie inne metody uzupełnienia danych możemy zastosować?
2. co gdy mamy dwa benchmarki, które chcielibyśmy połączyć?
3. czy zastosowanie PCA poprawia wyniki naszego modelu?
4. jak wpłynie na nasze modele bagging?
5. czy losowe przeszukiwanie przestrzeni parametrów lub metoda `irace` prowadzą do lepszych wyników?