xgboost

```r
library(mlr3)
library(mlr3learners)
library(mlr3filters)
library(mlr3pipelines)
library(mlr3tuning)
library(mlr3viz)
library(paradox)
library(tidyverse)
library(ggplot2)
library(patchwork)
library(future)
library(future.apply)
```

```r
future::plan("multiprocess")
```

xgboost Upload Rate Prediction

Reading the Data

First, the data has to be read from the .csv file:

```r
data_dir = "../datasets/"

dataset_ul = read_csv(
  str_c(data_dir, "dataset_ul.csv"),
  col_types = cols(scenario=col_factor())
) %>% rowid_to_column(var="row_id_original")

dataset_ul_prediction = dataset_ul %>% select(
  row_id_original,
  scenario,
  velocity_mps,
  acceleration_mpss,
  rsrp_dbm,
  rsrq_db,
  rssnr_db,
  cqi,
  ta,
  payload_mb,
  f_mhz,
  throughput_mbits
)

# remove missing values
dataset_ul_prediction = dataset_ul_prediction %>% drop_na()
glimpse(dataset_ul_prediction)
```

```
## Rows: 6,168
## Columns: 12
## $ row_id_original   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15...
## $ scenario          <fct> campus, campus, campus, campus, campus, campus, c...
## $ velocity_mps      <dbl> 11.80, 11.83, 11.70, 11.45, 11.49, 7.93, 8.15, 8....
## $ acceleration_mpss <dbl> 0.13, 0.03, 0.06, -0.32, -0.26, 0.23, 0.24, 0.32,...
## $ rsrp_dbm          <dbl> -99, -85, -121, -84, -97, -96, -74, -108, -111, -...
## $ rsrq_db           <dbl> -9, -5, -15, -6, -12, -12, -5, -9, -13, -11, -6, ...
## $ rssnr_db          <dbl> -1, 22, -8, 11, -2, 5, 29, 2, 6, 11, 13, 16, -3, ...
## $ cqi               <dbl> 8, 10, 4, 13, 9, 5, 15, 2, 6, 15, 12, 9, 6, 11, 1...
## $ ta                <dbl> 9, 7, 63, 4, 7, 7, 4, 21, 16, 7, 4, 4, 7, 16, 4, ...
## $ payload_mb        <dbl> 1.0, 4.0, 6.0, 2.0, 6.0, 5.0, 4.0, 10.0, 0.1, 7.0...
## $ f_mhz             <dbl> 1750, 1720, 1770, 1720, 1750, 1750, 1720, 1770, 1...
## $ throughput_mbits  <dbl> 4.66, 24.52, 1.29, 14.86, 3.97, 6.52, 16.27, 3.18...
```

Next, a prediction task has to be created to work with mlr3. The goal is to predict the variable `throughput_mbits`.

```r
task = TaskRegr$new(
  id = "ul_prediction",
  backend = dataset_ul_prediction,
  target = "throughput_mbits"
)
task$col_roles$name = "row_id_original"
task$col_roles$feature = setdiff(task$col_roles$feature, "row_id_original")
task
```

```
## <TaskRegr:ul_prediction> (6168 x 11)
## * Target: throughput_mbits
## * Properties: -
## * Features (10):
##   - dbl (9): acceleration_mpss, cqi, f_mhz, payload_mb, rsrp_dbm,
##     rsrq_db, rssnr_db, ta, velocity_mps
##   - fct (1): scenario
```

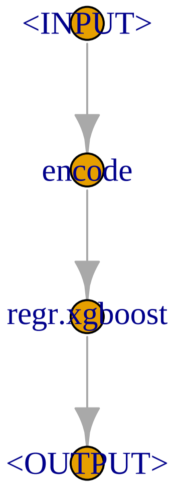
Creating the Prediction Pipeline

The next step is to create the prediction pipeline. It will consist of a factor encoding followed by the xgboost learner. The method for factor encoding is one-hot encoding. We put the pipeline creation inside of a function so that we can easily create untrained copies of the pipeline later when we need them.

```r
make_pipeline = function() {
  factor_encoding = po(
    "encode",
    method = "one-hot",
    affect_columns = selector_type("factor")
  )
  xgboost = lrn("regr.xgboost")
  pipe = factor_encoding %>>% xgboost
  return(pipe)
}
```

Let's see what the pipeline looks like:

```r
pipe = make_pipeline()
pipe$plot(html=FALSE)
```



Now the pipeline has to be converted to a learner so it can be used during training and prediction:

```
learner = GraphLearner$new(pipe)
```

Parameter Tuning

First, we have to define the set of parameters we use to tune the learner:

```
parameters = ParamSet$new(list(
  ParamInt$new("regr.xgboost.nrounds", lower=10, upper=500),
  ParamDbl$new("regr.xgboost.gamma", lower=0, upper=10),
  ParamInt$new("regr.xgboost.max_depth", lower=1, upper=10),
  ParamDbl$new("regr.xgboost.min_child_weight", lower=1, upper=100)
))
```

Next, we specify the tuning algorithm. For now, we use grid search:

```
tuner = tnr("grid_search", resolution=10)
```

Now all that is left is putting together the parts using the AutoTuner class. The resulting object is a learner that can automatically tune its parameters using the algorithm we specified.

```
tuned_learner = AutoTuner$new(
  learner = learner,
  resampling = rsmp("cv", folds = 5),
  measure = msr("regr.mae"),
  search_space = parameters,
  terminator = trm("evals", n_evals=10),
  tuner = tuner
)
```

Training and Benchmarking

Create a default learner to compare it to the tuned learner:

```
make_default_learner = function() {
  learner_default = GraphLearner$new(
    make_pipeline()
  )
  learner_default$param_set$values = mlr3misc::insert_named(
    learner_default$param_set$values,
    list(regr.xgboost.nrounds=100)
  )
  return(learner_default)
}
default_learner = make_default_learner()
```

```
benchmark_design = benchmark_grid(
  tasks = task,
  learners = list(tuned_learner, default_learner),
  resamplings = rsmp("cv", folds=5)
)
print(benchmark_design)
```

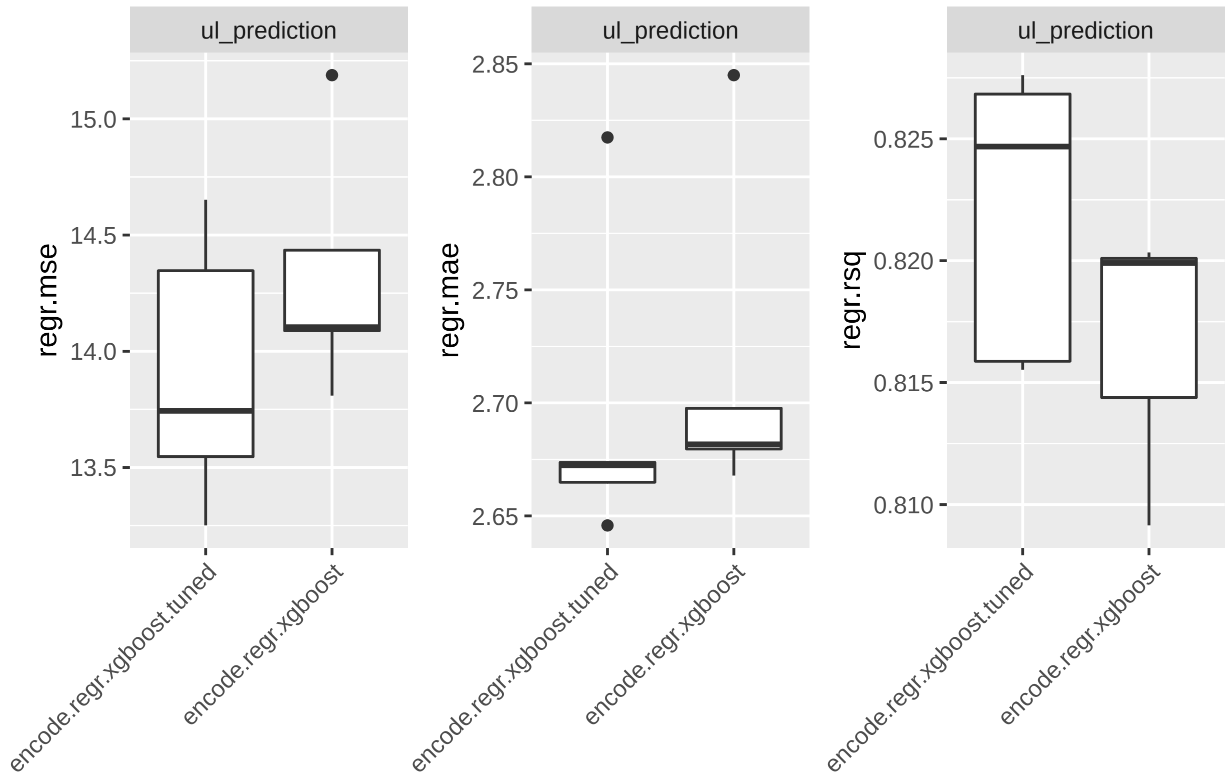
```
##               task            learner         resampling
## 1: <TaskRegr[42]>    <AutoTuner[37]> <ResamplingCV[19]>
## 2: <TaskRegr[42]> <GraphLearner[32]> <ResamplingCV[19]>
```

```
benchmark_result = benchmark(benchmark_design)
result_table = benchmark_result$aggregate(list(
  msr("regr.mse"),
  msr("regr.mae"),
  msr("regr.rsq")
))
```

```
knitr::kable(result_table[, c("learner_id", "regr.mse", "regr.mae", "regr.rsq")])
```

learner_id	regr.mse	regr.mae	regr.rsq
encode.regr.xgboost.tuned	13.90765	2.694845	0.8221083
encode.regr.xgboost	14.32468	2.714361	0.8167758

```
plot_mse = autoplot(benchmark_result, measure=msr("regr.mse")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
plot_mae = autoplot(benchmark_result, measure=msr("regr.mae")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
plot_rsq = autoplot(benchmark_result, measure=msr("regr.rsq")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
plot_mse + plot_mae + plot_rsq
```



Results

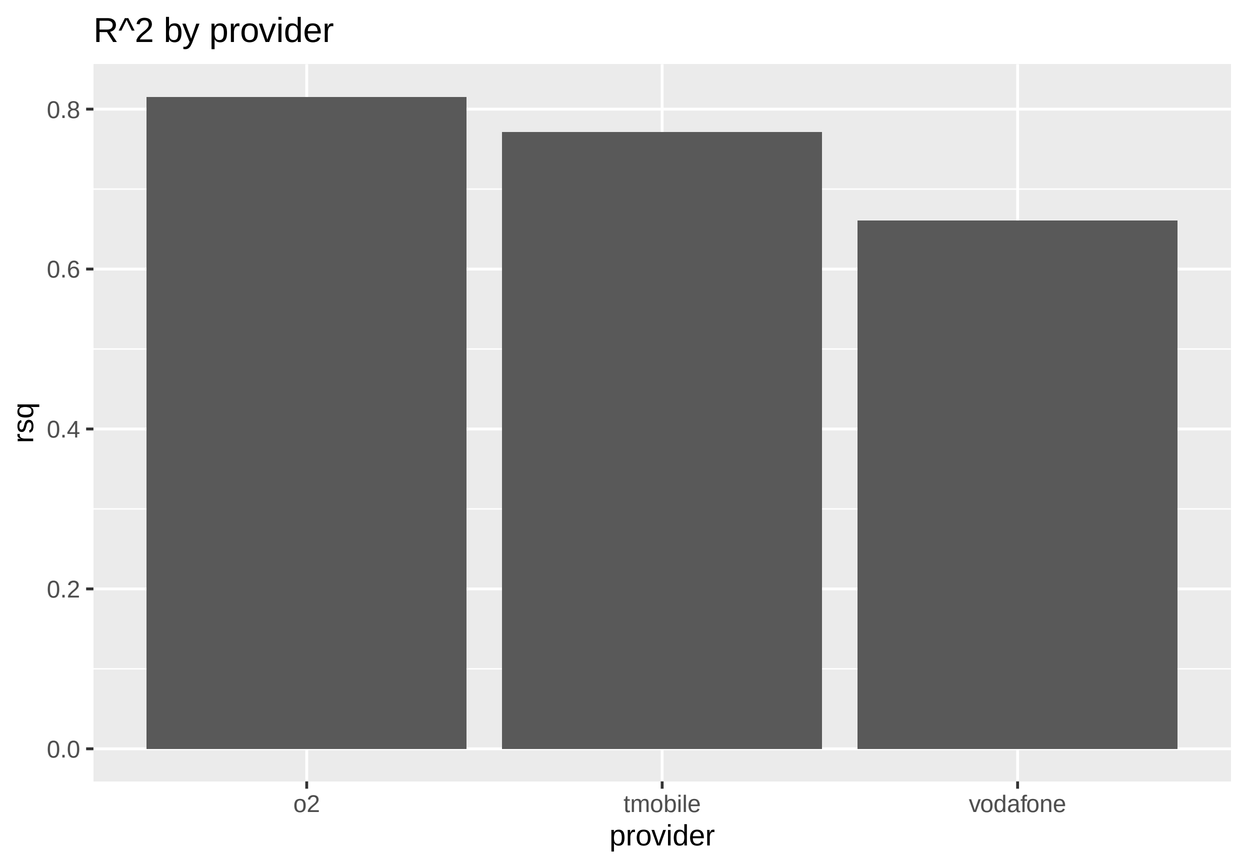
```
predictions = as_tibble(as.data.table(result_table[[1, "resample_result"]]$prediction())) %>%
  inner_join(as_tibble(task$row_names), by="row_id")
predictions_provider_scenario = predictions %>%
  inner_join(dataset_ul, by=c("row_name"="row_id_original")) %>%
  select(
    row_id,
    truth,
    response,
    provider,
    scenario
  )
predictions_provider_scenario
```

```
## # A tibble: 6,168 x 5
##    row_id truth response provider scenario
##     <int> <dbl>    <dbl> <chr>    <fct>
##  1      4 14.9     17.2  tmobile  campus
##  2      8  3.18     4.48 vodafone campus
##  3      9  0.05     2.02 vodafone campus
##  4     11 12.7     21.1  tmobile  campus
##  5     15 13.1     14.1  tmobile  campus
##  6     16  1.04     4.97 o2       campus
##  7     17  8.79     7.98 vodafone campus
##  8     20 15.9     18.7  tmobile  campus
##  9     24  2.18   -0.193 o2       campus
## 10     32 22.0    23.9   tmobile  campus
## # ... with 6,158 more rows
```

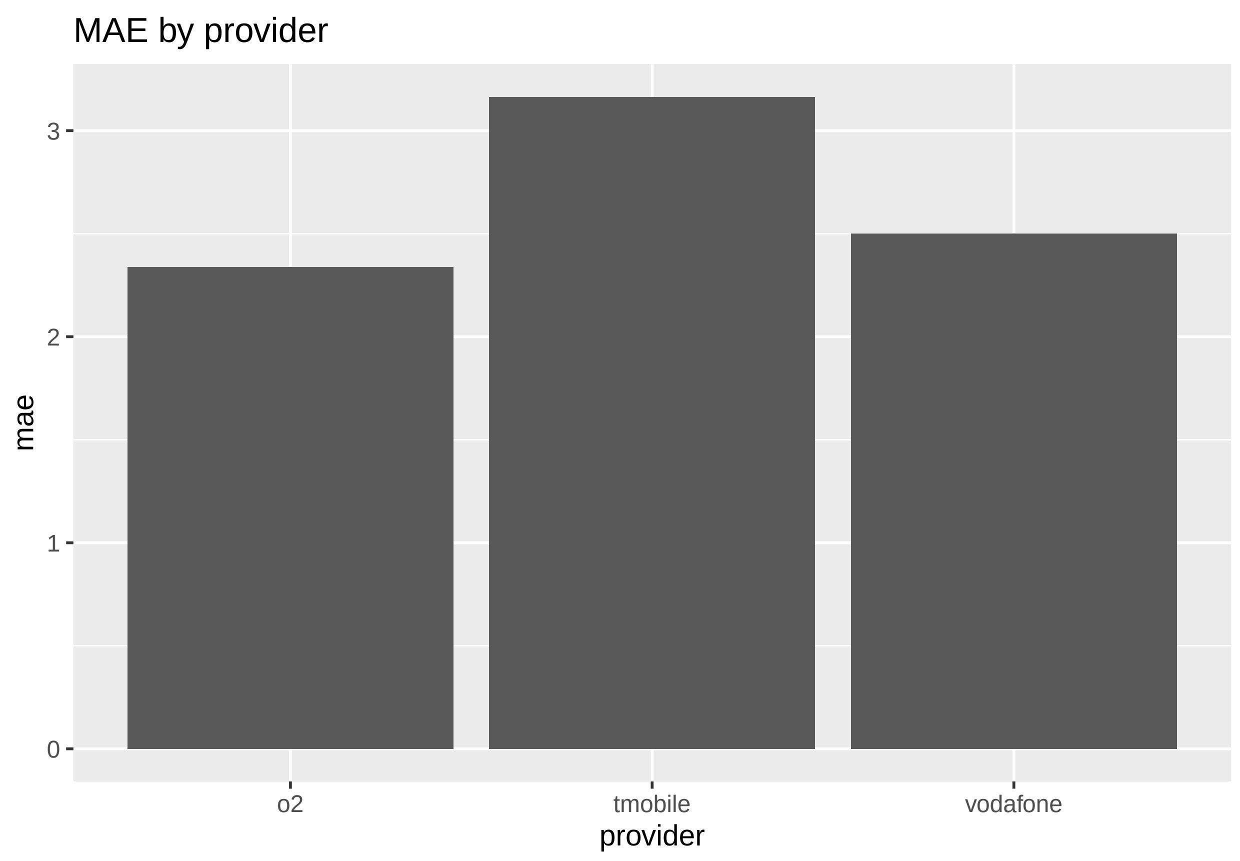
```r
get_measure = function(measure, providers, scenarios) {
  subset = predictions_provider_scenario %>%
    filter(provider %in% providers, scenario %in% scenarios)
  tmp_prediction = PredictionRegr$new(
    row_ids = subset$row_id,
    truth = subset$truth,
    response = subset$response
  )
  result = measure$score(tmp_prediction)
  return(result)
}
```

```r
providers = c("o2", "tmobile", "vodafone")
scenarios = scenarios=c("campus", "suburban", "highway", "urban")

results_by_provider = tibble(
  provider = providers,
  rsq = sapply(providers, function(p) {
    get_measure(msr("regr.rsq"), providers=p, scenarios=scenarios)
  }),
  mae = sapply(providers, function(p) {
    get_measure(msr("regr.mae"), providers=p, scenarios=scenarios)
  })
)

results_by_scenario = tibble(
  scenario = scenarios,
  rsq = sapply(scenarios, function(s) {
    get_measure(msr("regr.rsq"), providers=providers, scenarios=s)
  }),
  mae = sapply(scenarios, function(s) {
    get_measure(msr("regr.mae"), providers=providers, scenarios=s)
  })
)
```

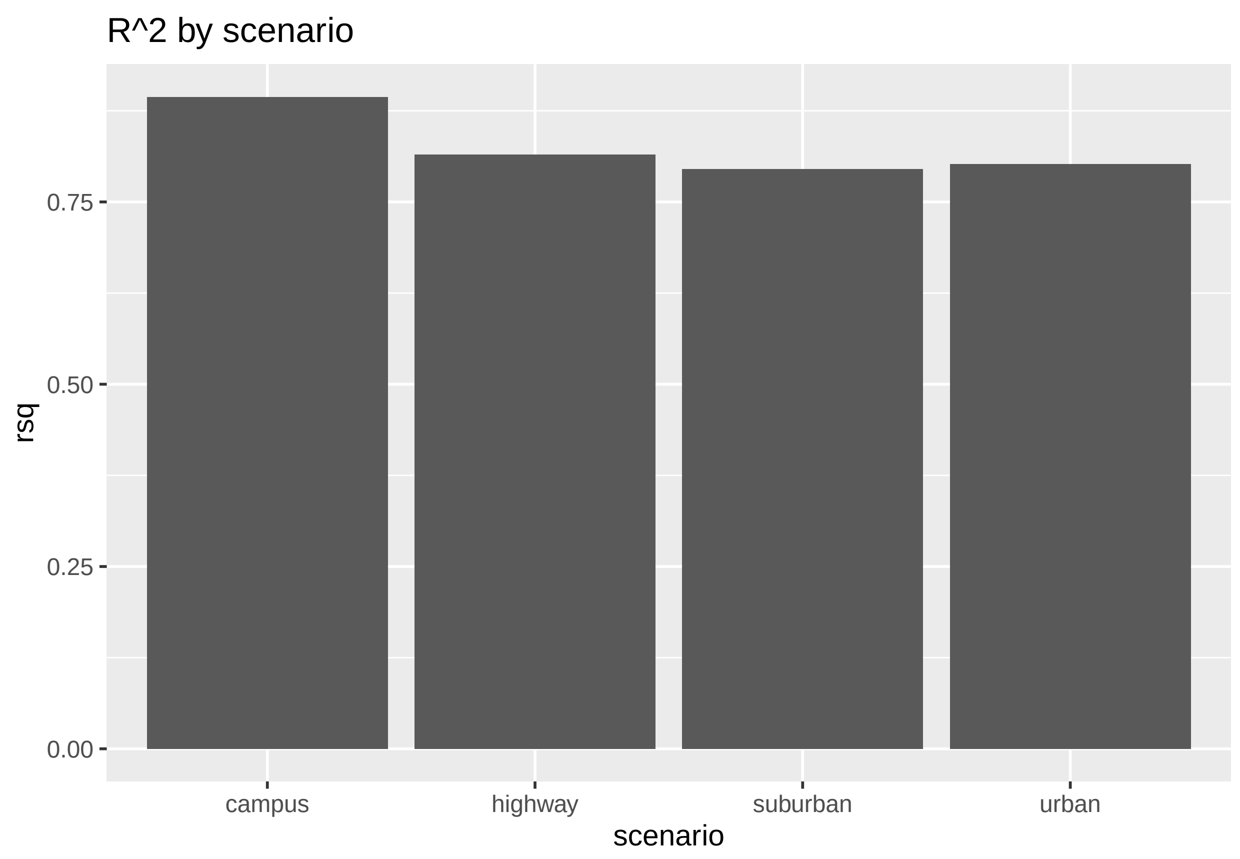
```r
ggplot(results_by_provider) +
  geom_bar(aes(x=provider, y=rsq), stat="identity") +
  ggtitle("R^2 by provider")
```



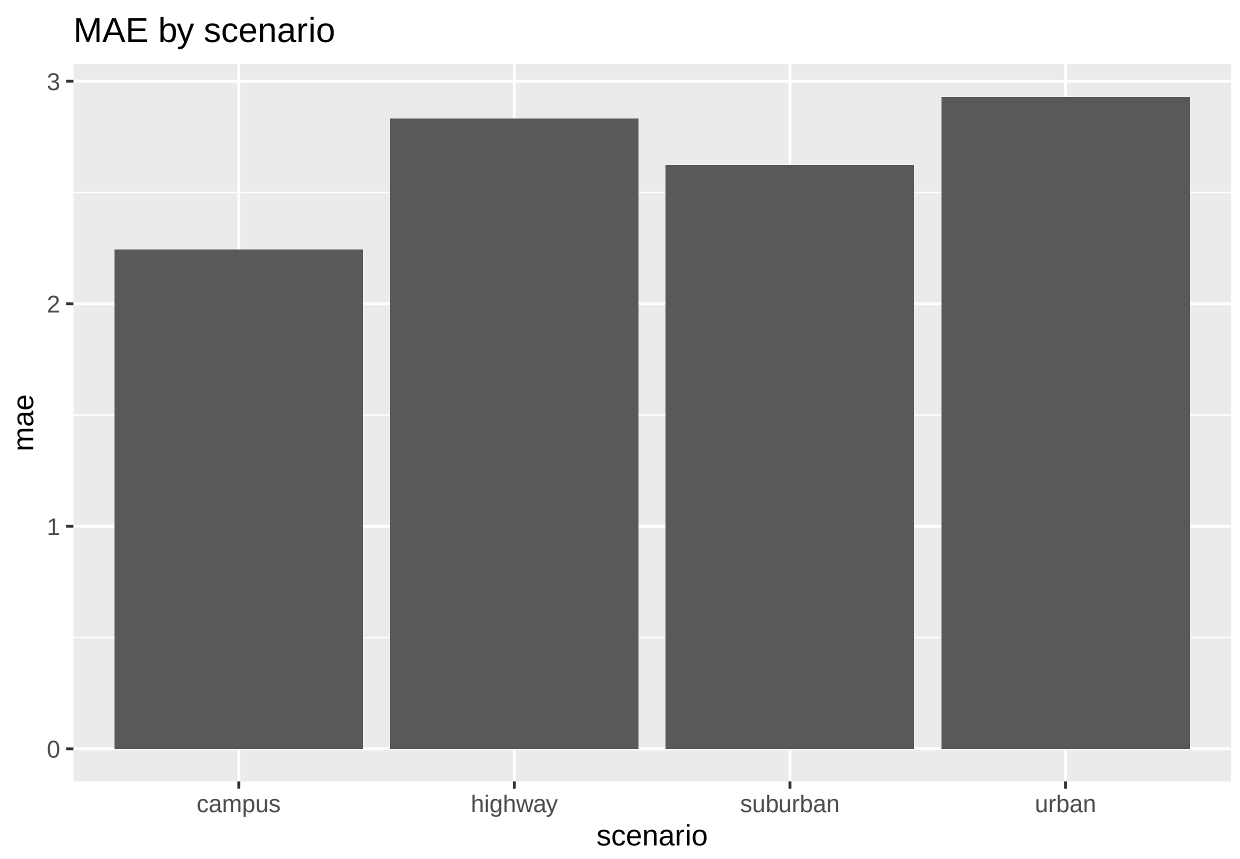
```r
ggplot(results_by_provider) +
  geom_bar(aes(x=provider, y=mae), stat="identity") +
  ggtitle("MAE by provider")
```



```
ggplot(results_by_scenario) +
  geom_bar(aes(x=scenario, y=rsq), stat="identity") +
  ggtitle("R^2 by scenario")
```



```
ggplot(results_by_scenario) +
  geom_bar(aes(x=scenario, y=mae), stat="identity") +
  ggtitle("MAE by scenario")
```



Feature Importance

Create an untrained learner with default parameters:

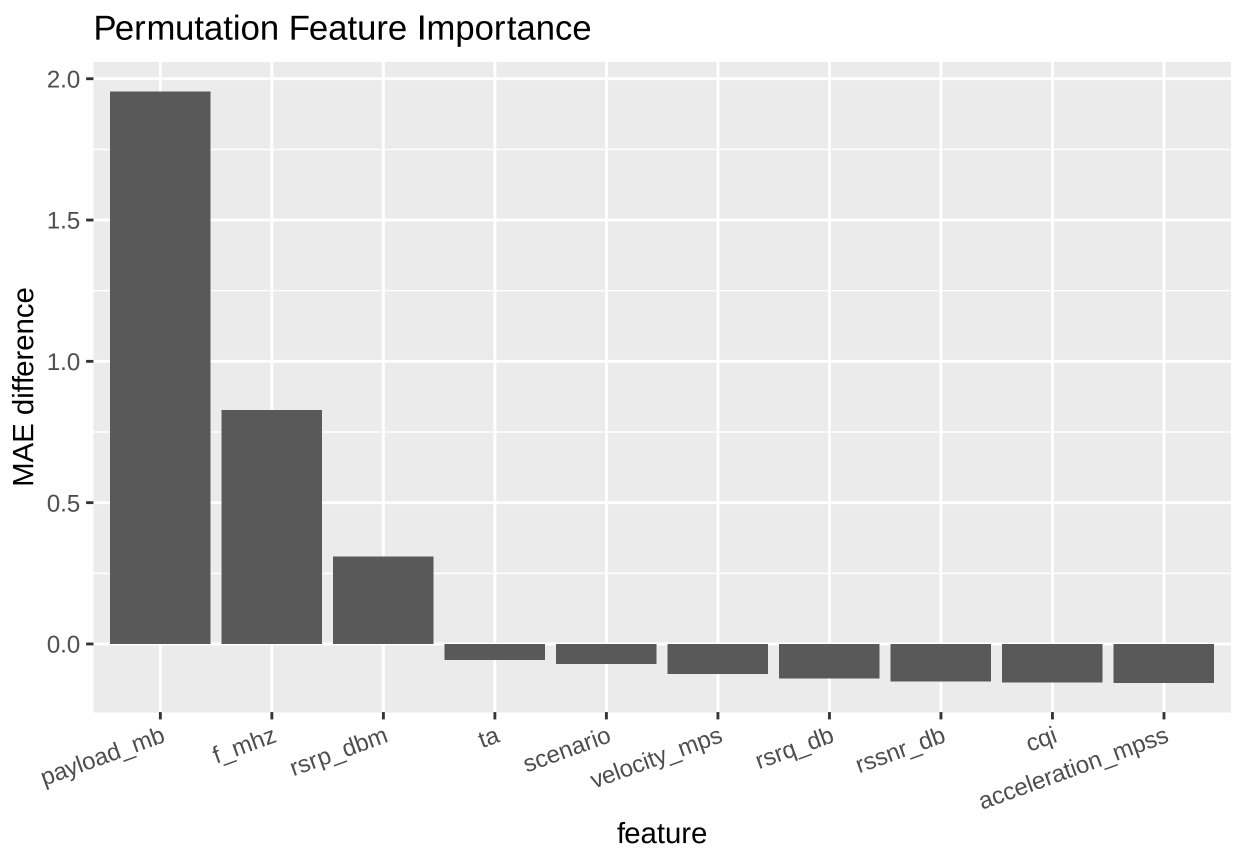
```
learner_default = make_default_learner()
```

```
filter_permutation = flt("permutation",
  learner = learner_default,
  resampling = rsmp("holdout", ratio=0.8),
  measure = msr("regr.mae"),
  standardize = TRUE,
  nmc=10
)
filter_permutation$calculate(task)
filter_permutation_results = as.data.table(filter_permutation)
```

```
filter_permutation_results
```

```
##             feature       score
##  1:      payload_mb  1.95492289
##  2:           f_mhz  0.82739730
##  3:        rsrp_dbm  0.30877614
##  4:              ta -0.05511798
##  5:        scenario -0.07076114
##  6:    velocity_mps -0.10608673
##  7:         rsrq_db -0.12079802
##  8:        rssnr_db -0.13270731
##  9:                cqi -0.13571072
## 10: acceleration_mpss -0.13650178
```

```r
ggplot(filter_permutation_results) +
  geom_bar(aes(x = reorder(feature, -score), y = score), stat="identity") +
  xlab("feature") +
  ylab("MAE difference") +
  scale_x_discrete(guide = guide_axis(angle = 20)) +
  ggtitle("Permutation Feature Importance")
```


Permutation Feature Importance
MAE difference
payload_mb
f_mhz
rsrp_dbm
ta
scenario
velocity_mps
rsrq_db
rssnr_db
cqi
acceleration_mpss
feature