Exercise 2 – Regression

Introduction to Machine Learning

Hint: Useful libraries

R

```
# Consider the following libraries for this exercise sheet:

library(ggplot2)
library(mlr3verse)
library(mlr3learners)
library(mlr3viz)
library(quantreg)
```

Python

```
# Consider the following libraries for this exercise sheet:

# general
import numpy as np
import pandas as pd
import math
# plots
import matplotlib.pyplot as plt
import seaborn as sns
# sklearn
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
```

```python
from sklearn.metrics import mean_absolute_error
```

Exercise 1: HRO in coding frameworks

> Learning goals
>
> Translate lecture concepts to code

Throughout the lecture, we will frequently use the `R` package `mlr3`, resp. the `Python` package `sklearn`, and its descendants, providing an integrated ecosystem for all common machine learning tasks. Let's recap the HRO principle and see how it is reflected in either `mlr3` or `sklearn`. An overview of the most important objects and their usage, illustrated with numerous examples, can be found at the `mlr3` book and the `scikit` documentation.

How are the key concepts (i.e., hypothesis space, risk and optimization) you learned about in the lecture videos implemented?

Solution

R

```r
# H: First, initialize your learner.
# Before training learners just contain information on the functional form of f.
model <- lrn("regr.lm")
print(model)
x <- seq(0, 8, by = 0.01)
set.seed(42)
y <- -1 + 3 * x + rnorm(mean = 0, sd = 4, n = length(x))
dt <- data.frame(x = x, y = y)
task <- TaskRegr$new(id = "mytask", backend = dt, target = "y")
# R: `mlr3` relies on package-specific learning objectives.
# O: Optimization is triggered by `model$train()`, internally calling
# package-specific optimization procedures.
model$train(task)
sprintf("Model MSE: %.4f", model$predict_newdata(dt)$score())
```

```
<LearnerRegrLM:regr.lm>: Linear Model
* Model: -
* Parameters: list()
* Packages: mlr3, mlr3learners, stats
* Predict Types:  [response], se
* Feature Types: logical, integer, numeric, character, factor
* Properties: loglik, weights
```

'Model MSE: 15.1048'

Python

```python
# H: First, initialize your learner.
# Before training learners just contain information on the functional form of f.
model = LinearRegression(fit_intercept=True)
print(model)
x = np.arange(0, 8, 0.01)
np.random.seed(42)
y = -1 + 3 * x + np.random.normal(loc=0.0, scale=4, size=len(x))
# R: `sklearn` relies on package-specific learning objectives.
# O: Optimization is triggered by `model.fit()`, internally calling
# package-specific optimization procedures.
# within the function `model.fit()`:
model.fit(x.reshape(-1, 1), y) # reshape for one feature design matrix
mse = metrics.mean_squared_error(y, model.predict(x.reshape(-1, 1)))
print(f'Model MSE: {mse:.4f}')
```

```
LinearRegression()
Model MSE: 15.4618
```

Have a look at`mlr3::tsk("iris")` / `sklearn.datasets.load_iris`. What attributes does this object store?

Solution

R

```r
task_iris <- tsk("iris")
sprintf("Feature names: %s", task_iris$feature_names)
sprintf("Target name: %s", task_iris$target_names)
```

1. 'Feature names: Petal.Length'
2. 'Feature names: Petal.Width'
3. 'Feature names: Sepal.Length'
4. 'Feature names: Sepal.Width'

'Target name: Species'

Python

```python
iris = load_iris() # function to import iris as type "utils.Bunch" with sklearn
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Type of object iris:", type(iris))
print("Feature names:"), [print(f'{i}') for i in feature_names]
print("Target names:", target_names)
print("\nShape of X and y\n", X.shape, y.shape)
print("\nType of X and y\n", type(X), type(y))
```

```
Type of object iris: <class 'sklearn.utils._bunch.Bunch'>
Feature names:
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
Target names: ['setosa' 'versicolor' 'virginica']

Shape of X and y
 (150, 4) (150,)

Type of X and y
 <class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Instantiate a regression tree learner (`lrn("regr.rpart")` / `DecisionTreeRegressor`). What are the different settings for this learner?

Hint

R

`mlr3::mlr_learners$keys()` shows all available learners.

Python

Use `get_params()` to see all available settings.

Solution

R

```
# List available learners in base mlr3 package
head(mlr_learners$keys())

# Inspect regression tree learner
lrn("regr.rpart")

# List configurable hyperparameters
as.data.table(lrn("regr.rpart")$param_set)
```

1. 'classif.cv_glmnet'
2. 'classif.debug'
3. 'classif.featureless'
4. 'classif.glmnet'
5. 'classif.kknn'
6. 'classif.lda'

```
<LearnerRegrRpart:regr.rpart>: Regression Tree
* Model: -
* Parameters: xval=0
* Packages: mlr3, rpart
* Predict Types:  [response]
```

```
* Feature Types: logical, integer, numeric, factor, ordered
* Properties: importance, missings, selected_features, weights
```

A data.table: 10 × 11

id	class	lower	upper	levels	nlevels	is_bounded	special_vals	default	storage_type	tags
<chr>	<chr>	<dbl>	<dbl>	<list>	<dbl>	<lgl>	<list>	<list>	<chr>	<list>
cp	ParamDbl	0	1	NULL	Inf	TRUE	NULL	0.01	numeric	train
keep_model	ParamLgl	NA	NA	TRUE, FALSE	2	TRUE	NULL	FALSE	logical	train
maxcompete	ParamInt	0	Inf	NULL	Inf	FALSE	NULL	4	integer	train
maxdepth	ParamInt	1	30	NULL	30	TRUE	NULL	30	integer	train
maxsurrogate	ParamInt	0	Inf	NULL	Inf	FALSE	NULL	5	integer	train
minbucket	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	<environment: 0x561f4e4c8480>	integer	train
minsplit	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	20	integer	train
surrogatestyle	ParamInt	0	1	NULL	2	TRUE	NULL	0	integer	train
usesurrogate	ParamInt	0	2	NULL	3	TRUE	NULL	2	integer	train
xval	ParamInt	0	Inf	NULL	Inf	FALSE	NULL	10	integer	train

Python

```python
# Inspect regression tree learner
rtree = DecisionTreeRegressor() # default setting
print(rtree)

# List configurable hyperparameters
[print(f'{k}: {v}') for k, v in rtree.get_params().items()][0]
```

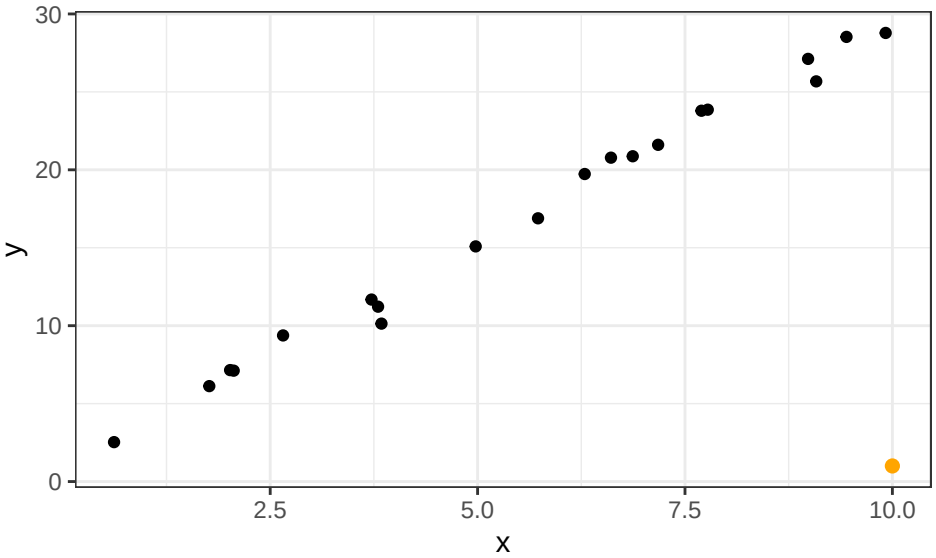
```
DecisionTreeRegressor()
ccp_alpha: 0.0
criterion: squared_error
max_depth: None
max_features: None
max_leaf_nodes: None
min_impurity_decrease: 0.0
min_samples_leaf: 1
min_samples_split: 2
min_weight_fraction_leaf: 0.0
random_state: None
splitter: best
```

Exercise 2: Loss functions for regression tasks

Learning goals

1. Assess how outliers affect models for different loss functions
2. Derive impact of a loss function from visual representation

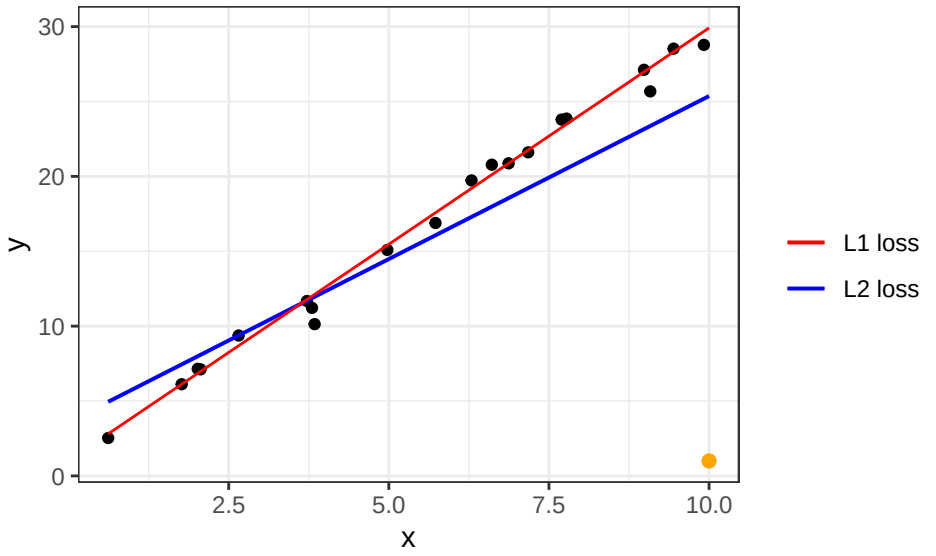
In this exercise, we will examine loss functions for regression tasks somewhat more in depth.

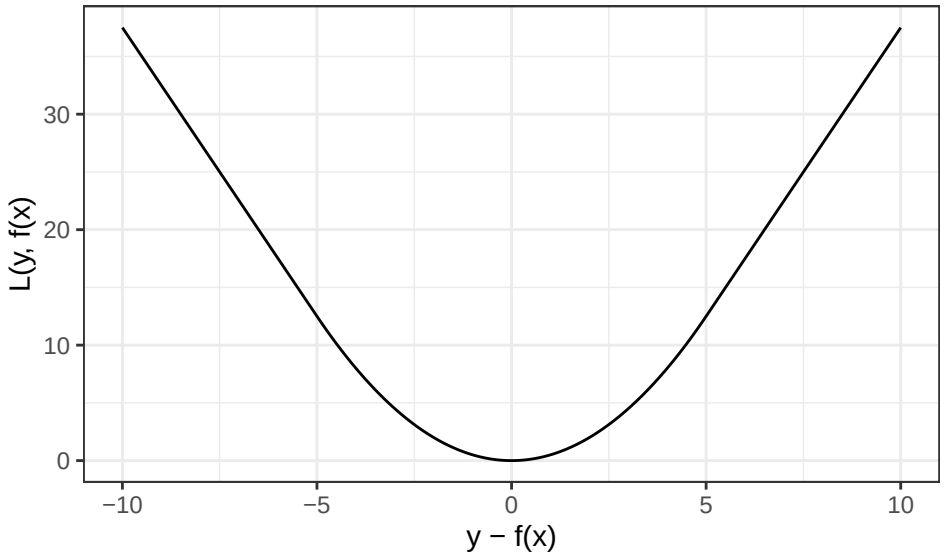


Consider the above linear regression task. How will the model parameters be affected by adding the new outlier point (orange) if you use $L1$ loss and $L2$ loss, respectively, in the empirical risk? (You do not need to actually compute the parameter values.)

Solution

$L2$ loss penalizes vertical distances to the regression line *quadratically*, while $L1$ only considers the *absolute* distance. As the outlier point lies pretty far from the remaining training data, it will have a large loss with $L2$, and the regression line will pivot to the bottom right to minimize the resulting empirical risk. A model trained with $L1$ loss is less susceptible to the outlier and will adjust only slightly to the new data.





The second plot visualizes another loss function popular in regression tasks, the so-called *Huber loss* (depending on $\epsilon > 0$; here: $\epsilon = 5$). Describe how the Huber loss deals with residuals as compared to $L1$ and $L2$ loss. Can you guess its definition?

Solution

The Huber loss combines the respective advantages of $L1$ and $L2$ loss: it is smooth and (once) differentiable like $L2$ but does not punish larger residuals as severely, leading to more

robustness. It is simply a (weighted) piecewise combination of both losses, where ϵ marks where $L2$ transits to $L1$ loss. The exact definition is:

$$L\left(y, f(\mathbf{x})\right) = \begin{cases} \frac{1}{2}(y - f(\mathbf{x}))^2 & \text{if } |y - f(\mathbf{x})| \leq \epsilon \\ \epsilon|y - f(\mathbf{x})| - \frac{1}{2}\epsilon^2 & \text{otherwise} \end{cases}, \quad \epsilon > 0$$

In the plot we can see how the parabolic shape of the loss around 0 evolves into an absolute-value function at $|y - f(\mathbf{x})| > \epsilon = 5$.

Exercise 3: Polynomial regression

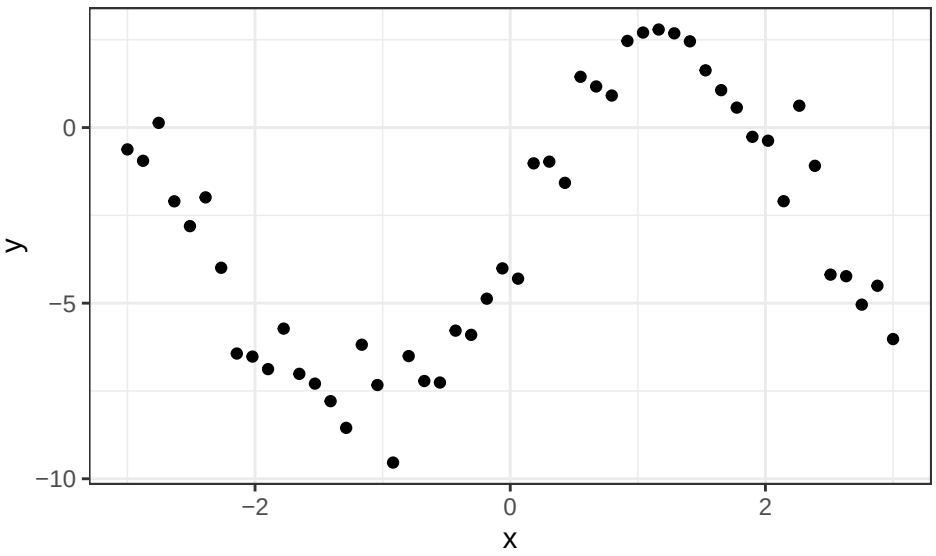
Learning goals

1. Express HRO components for polynomial regression
2. Derive gradient update for optimization
3. Analyze and discuss learner flexibility

Assume the following (noisy) data-generating process from which we have observed 50 realizations:

$$y = -3 + 5 \cdot \sin(0.4\pi x) + \epsilon$$

with $\epsilon \sim \mathcal{N}(0, 1)$.



We decide to model the data with a cubic polynomial (including intercept term). State the corresponding hypothesis space.

Solution

Cubic means degree 3, so our hypothesis space will look as follows:

$$\mathcal{H} = \{f(\mathbf{x} \mid \theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \mid (\theta_0, \theta_1, \theta_2, \theta_3)^\top \in \mathbb{R}^4\}$$

State the empirical risk w.r.t. θ for a member of the hypothesis space. Use $L2$ loss and be as explicit as possible.

Solution

The empirical risk is:

$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^{50} \left(y^{(i)} - \left[\theta_0 + \theta_1 x^{(i)} + \theta_2 \left(x^{(i)}\right)^2 + \theta_3 \left(x^{(i)}\right)^3\right]\right)^2$$

Only for lecture group A

We can minimize this risk using gradient descent. Derive the gradient of the empirical risk w.r.t θ.

Solution

We can find the gradient just as we did for an intermediate result when we derived the least-squares estimator:

$$\begin{aligned}
\nabla_\theta \mathcal{R}_{\text{emp}}(\theta) &= \frac{\partial}{\partial \theta} \|\mathbf{y} - \mathbf{X}\theta\|_2^2 \\
&= \frac{\partial}{\partial \theta} \left((\mathbf{y} - \mathbf{X}\theta)^\top (\mathbf{y} - \mathbf{X}\theta)\right) \\
&= -2\mathbf{y}^\top \mathbf{X} + 2\theta^\top \mathbf{X}^\top \mathbf{X} \\
&= 2 \cdot \left(-\mathbf{y}^\top \mathbf{X} + \theta^\top \mathbf{X}^\top \mathbf{X}\right)
\end{aligned}$$

Only for lecture group A

Using the result for the gradient, explain how to update the current parameter $\theta^{[t]}$ in a step of gradient descent.

Solution

Recall that the idea of gradient descent (*descent*!) is to traverse the risk surface in the direction of the *negative* gradient as we are in search for the minimum. Therefore, we will update our current parameter set $\theta^{[t]}$ with the negative gradient of the current empirical risk w.r.t. θ, scaled by learning rate (or step size) α:

$$\theta^{[t+1]} = \theta^{[t]} - \alpha \cdot \nabla_\theta \mathcal{R}_{\text{emp}}(\theta^{[t]}).$$

What actually happens here: we update each component of our current parameter vector $\theta^{[t]}$ in the *direction* of the negative gradient, i.e., following the steepest downward slope, and also by an *amount* that depends on the value of the gradient.

In order to see what that means it is helpful to recall that the gradient $\nabla_\theta \mathcal{R}_{\text{emp}}(\theta)$ tells us about the effect (infinitesimally small) changes in θ have on $\mathcal{R}_{\text{emp}}(\theta)$. Therefore, gradient updates focus on influential components, and we proceed more quickly along the important dimensions.

You will not be able to fit the data perfectly with a cubic polynomial. Describe the advantages and disadvantages that a more flexible model class would have. Would you opt for a more flexible learner?

Solution

We see that, for example, the first model in exercise b) fits the data fairly well but not perfectly. Choosing a more flexible function (a polynomial of higher degree or a function from an entirely different, more complex, model class) might be advantageous:

- We would be able to trace the observations more closely if our function were less smooth, and thus reduce empirical risk. On the other hand, flexibility also has drawbacks:
- Flexible model classes often have more parameters, making training harder.
- We might run into a phenomenon called *overfitting*. Recall that our ultimate goal is to make predictions on *new* observations. However, fitting every quirk of the training observations – possibly caused by imprecise measurement or other factors of randomness/error – will not generalize so well to new data.

In the end, we need to balance model fit and generalization. We will discuss the choice of hypotheses quite a lot since it is one of the most crucial design decisions in machine learning.

Exercise 4: Predicting `abalone`

> Learning goals
>
> 1. Implement regression model
> 2. Analyze basic regression fit

We want to predict the age of an abalone using its longest shell measurement and its weight. The `abalone` data can be found here: https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data.

Prepare the data as follows:

R

```r
# Download data
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
abalone <- read.table(url, sep = ",", row.names = NULL)
colnames(abalone) <- c(
  "sex", "longest_shell", "diameter", "height", "whole_weight",
  "shucked_weight", "visceral_weight", "shell_weight", "rings"
)

# Reduce to relevant columns
abalone <- abalone[, c("longest_shell", "whole_weight", "rings")]
```

Python

```python
# load data from url

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
abalone = pd.read_csv(
    url,
    sep=',',
    names=[
        'sex',
```

```
        "longest_shell",
        "diameter",
        "height",
        "whole_weight",
        "shucked_weight",
        "visceral_weight",
        "shell_weight",
        "rings"
    ]
)

abalone = abalone[['longest_shell', 'whole_weight', 'rings']]
print(abalone.head)
```

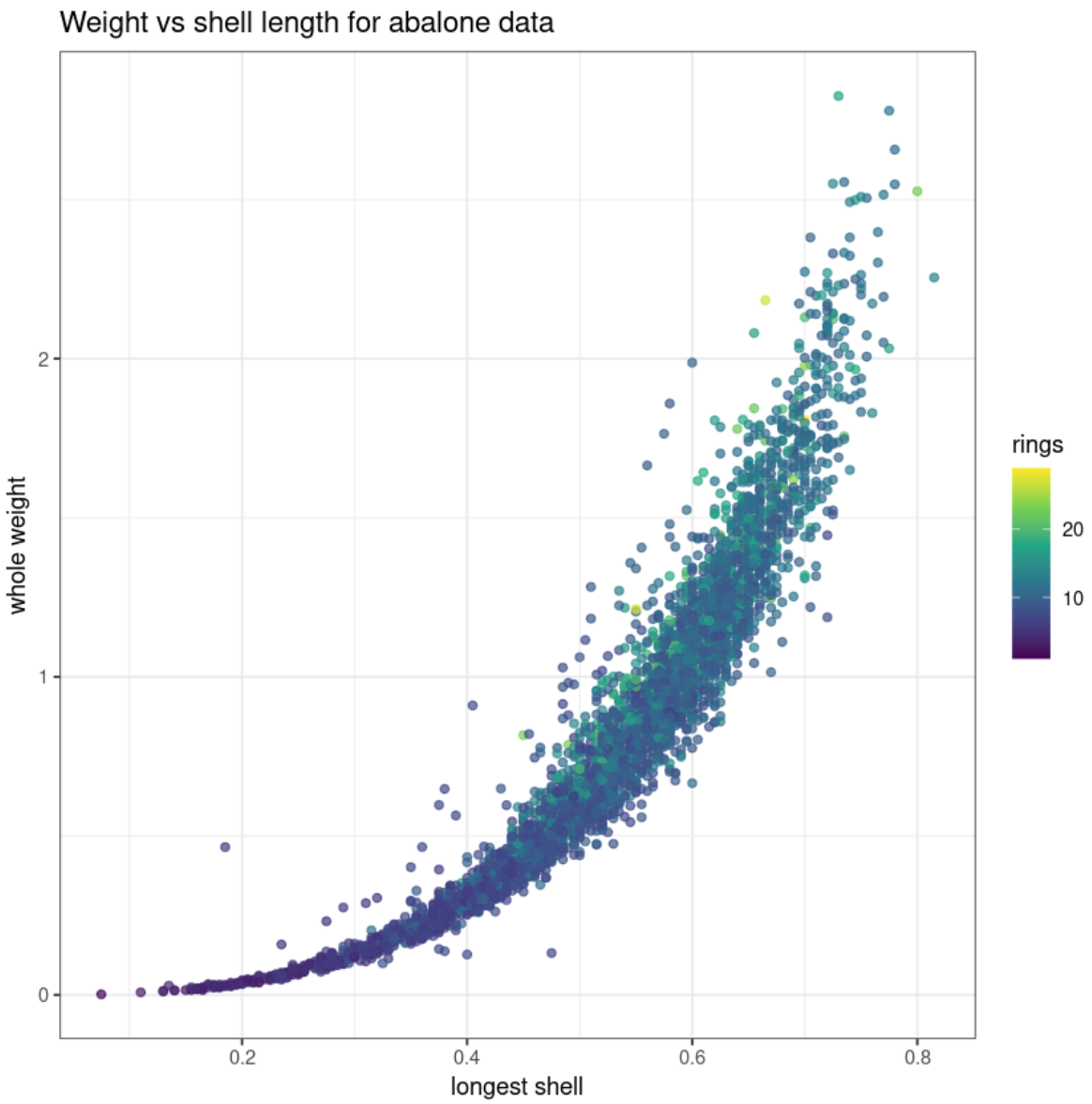
```
<bound method NDFrame.head of       longest_shell  whole_weight  rings
0              0.455        0.5140     15
1              0.350        0.2255      7
2              0.530        0.6770      9
3              0.440        0.5160     10
4              0.330        0.2050      7
...              ...           ...    ...
4172           0.565        0.8870     11
4173           0.590        0.9660     10
4174           0.600        1.1760      9
4175           0.625        1.0945     10
4176           0.710        1.9485     12

[4177 rows x 3 columns]>
```

Plot `LongestShell` and `WholeWeight` on the x- and y-axis, respectively, and color points according to `Rings`.

Solution

R

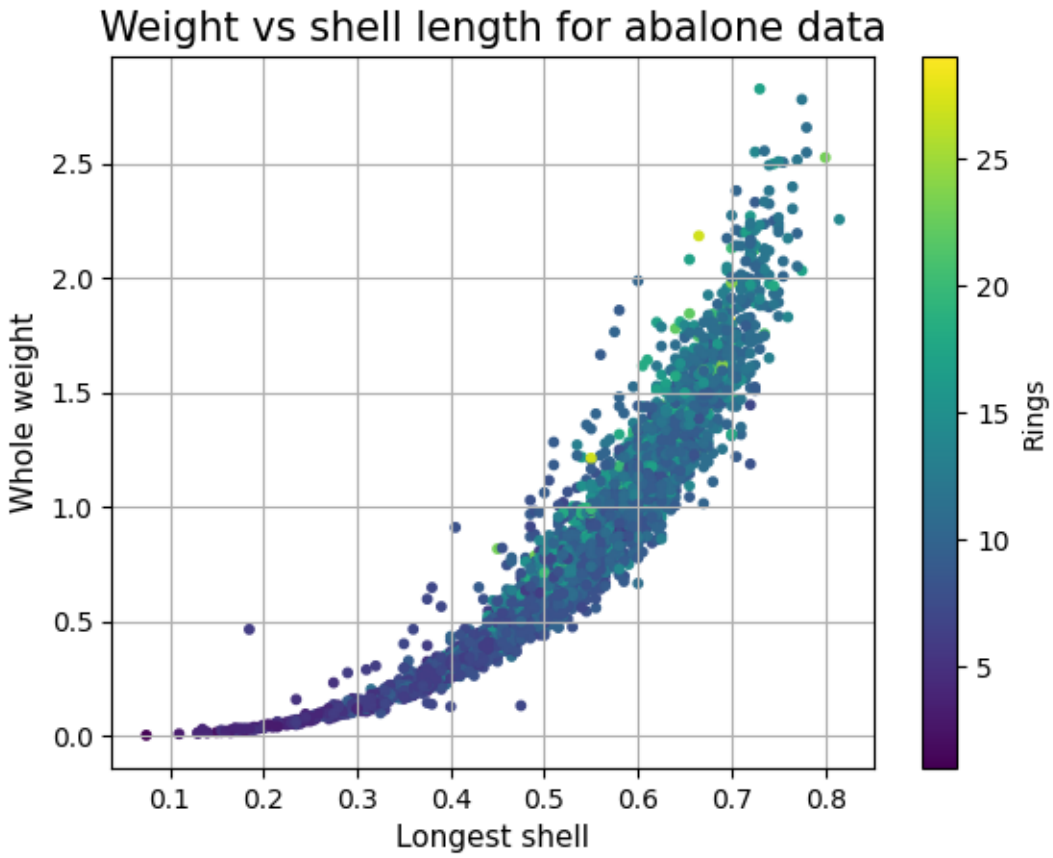
```r
# Plot weight vs shell length
plot <- ggplot(
  abalone,
  aes(x = longest_shell, y = whole_weight, col = rings)) +
  geom_point(alpha = 0.7) +
  scale_color_viridis_c() +
  theme_bw() +
  labs(
    x = "Longest shell",
    y = "Whole weight",
    title = "Weight vs shell length for abalone data"
  )
print(plot)
```



Python

```
plt.grid(True)
plt.scatter(
    abalone.longest_shell,
    abalone.whole_weight,
    s=10,
```

```
    c=abalone.rings,
    cmap = 'viridis'
)
plt.colorbar(label = 'Rings') # add color bar
# title & label axes
plt.title('Weight vs shell length for abalone data', size=15)
plt.xlabel('Longest shell', size=11)
plt.ylabel('Whole weight', size=11)
plt.show()
```



We see that weight scales exponentially with shell length and that larger/heavier animals tend to have more rings.

Using `mlr3`/`sklearn`, fit a linear regression model to the data.

Solution

R

```r
# Specify regression task
task_abalone <- TaskRegr$new(
  id = "abalone", backend = abalone, target = "rings"
)
task_abalone

# Set up LM, train (by default, the target will be regressed on all features,
# i.e., target ~ .)
learner_lm <- mlr3::lrn("regr.lm")
print("Model before training:")
learner_lm$model

# Train and predict
learner_lm$train(task_abalone)
print("Model after training:")
learner_lm$model
pred_lm <- learner_lm$predict(task_abalone)

# Inspect predictions
print("Predictions:")
pred_lm
```

```
<TaskRegr:abalone> (4177 x 3)
* Target: rings
* Properties: -
* Features (2):
  - dbl (2): longest_shell, whole_weight

[1] "Model before training:"
[1] "Model after training:"
[1] "Predictions:"

NULL


Call:
```

```
stats::lm(formula = task$formula(), data = task$data())

Coefficients:
  (Intercept)  longest_shell   whole_weight
        3.431         10.582          1.155


<PredictionRegr> for 4177 observations:
    row_ids truth  response
          1    15  8.840042
          2     7  7.395659
          3     9  9.821995
---
       4175     9 11.139128
       4176    10 11.309553
       4177    12 13.195460
```

Python

```
x_lm = abalone.iloc[:, 0:2].values
y_lm = abalone.rings
lm = LinearRegression().fit(x_lm,y_lm)
pred_lm = lm.predict(x_lm)
results_dic = {'prediction' : pred_lm, 'truth': y_lm}
results = pd.DataFrame(results_dic)
results.head()
```

	prediction	truth
0	8.840042	15
1	7.395659	7
2	9.821995	9
3	8.683616	10
4	7.160333	7

Compare the fitted and observed targets visually.

R Hint

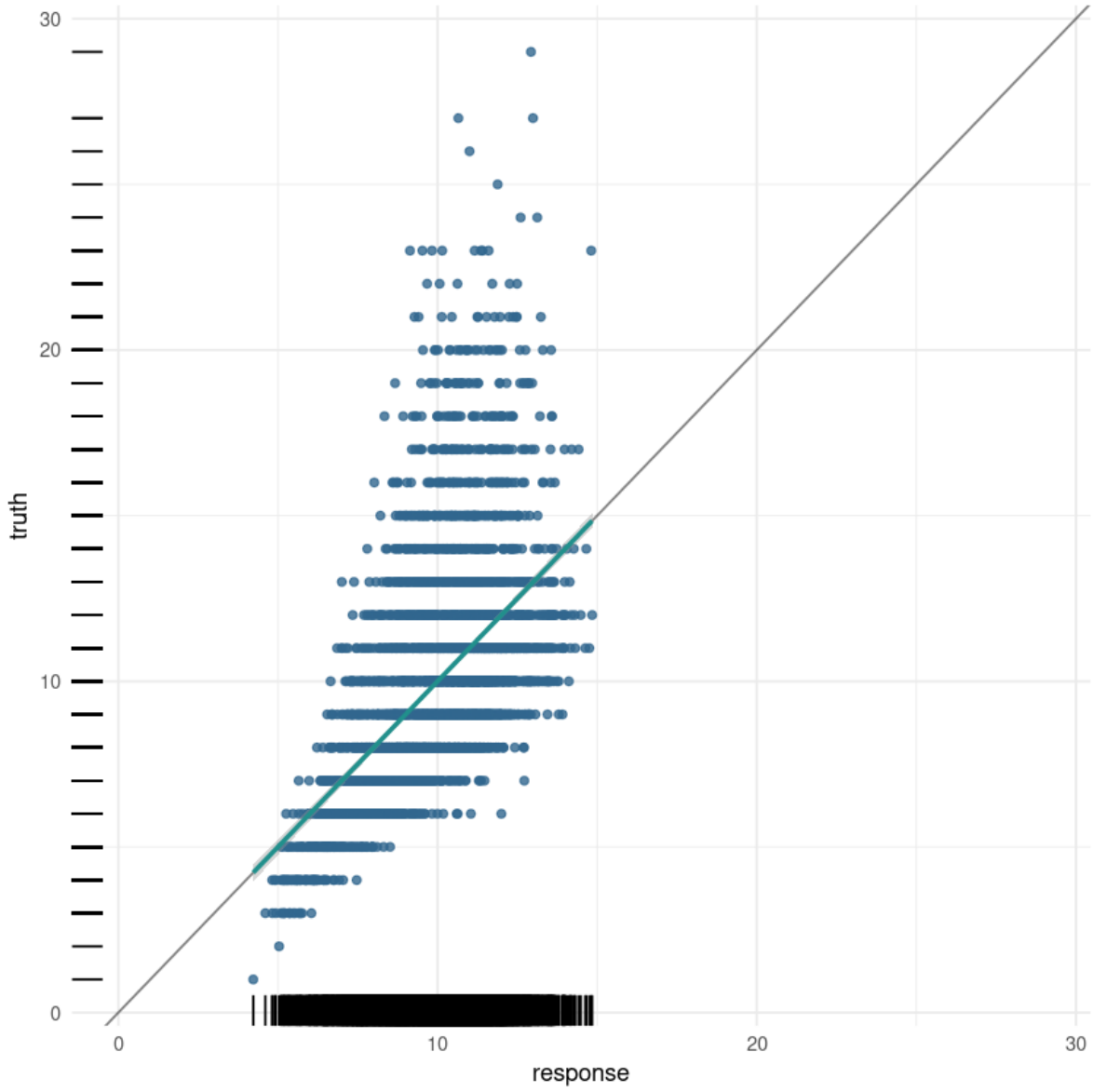
R

Use `$autoplot()` from `mlr3viz`.

Solution

R

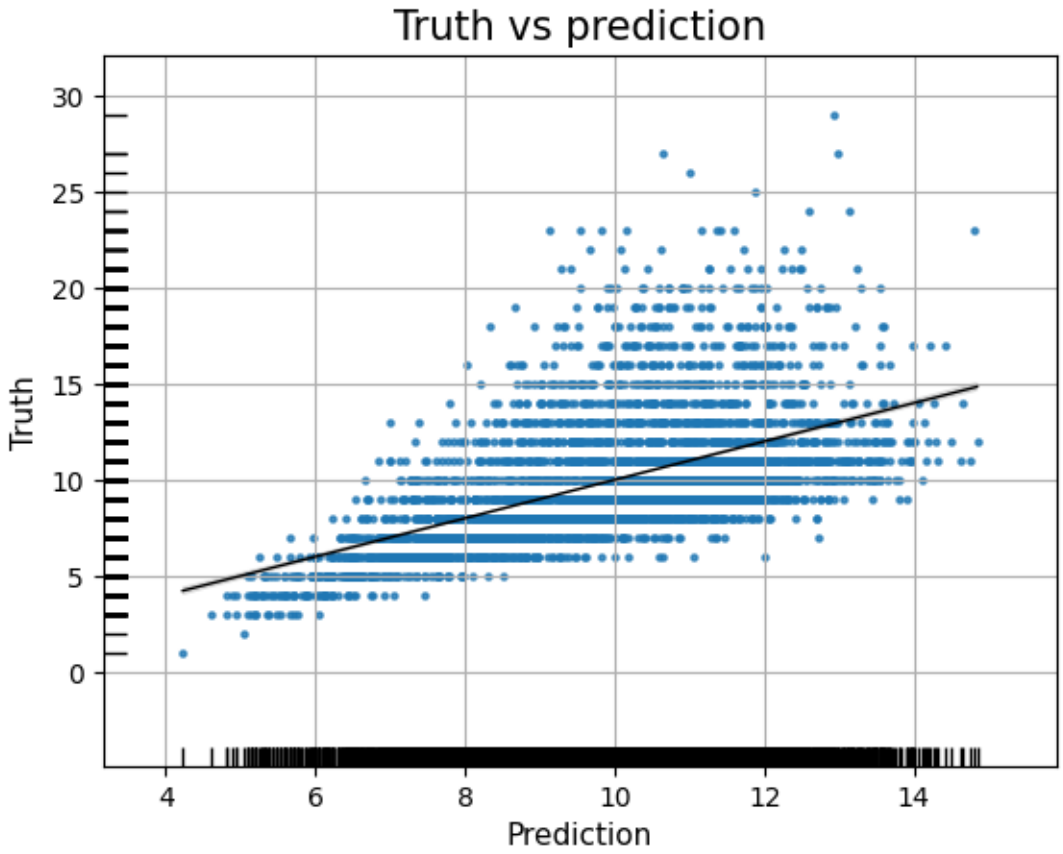
```
# Get nice visualization with a one-liner
mlr3viz::autoplot(pred_lm) +
  xlim(c(0, max(abalone$rings)))
```



Python

```
plt.grid(True)
sns.regplot(
    x=pred_lm,
    y=y_lm,
    ci=95,
```

```
    scatter_kws={'s': 5},
    line_kws={"color": "black", 'linewidth': 1}
)
sns.rugplot(x=pred_lm, y=y_lm, height=0.025, color='k')
# title & label axes
plt.title('Truth vs prediction', size=15)
plt.xlabel('Prediction', size=11)
plt.ylabel('Truth', size=11)
plt.show()
```



We see a scatterplot of prediction vs true values, where the small bars along the axes (a so-called rugplot) indicate the number of observations that fall into this area. As we might have suspected from the first plot, the underlying relationship is not exactly linear (ideally, all points and the resulting line should lie on the diagonal). With a linear model we tend to underestimate the response.

Assess the model's training loss in terms of MAE.

Hint

R

Call `$score()`, which accepts different `mlr_measures`, on the prediction object.

Python

Call `from sklearn.metrics import mean_absolute_error`.

Solution

R

```
# Define MAE metric
mae <- msr("regr.mae")

# Assess performance (MSE by default)
round(pred_lm$score(), 4)
round(pred_lm$score(mae), 4)
```

regr.mse: 7.1255

regr.mae: 1.9507

Python

```
mae = mean_absolute_error(pred_lm, y_lm)
print(f'{mae:.4f}')
```

```
1.9507
```