Cyber-Physical Systems
in Mechanical Engineering TU Berlin

Applied Machine Learning in Engineering

Lecture 02 winter term 2023/24

Prof. Merten Stender

Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin

www.tu.berlin/cpsme
merten.stender@tu-berlin.de

Organizational Note

- MOSES description has been changed w.r.t. examination
- Final examination will be a written exam (digital, at TU Berlin)
- Dates: 21.02.24 (11am); 02.04.24 (02pm)

→ Course is open to **everyone** to attend and take the exam

→ No homework assignments, no oral exam

Module completion

Grading

graded

Type of exam

Written exam

Duration/Extent

(Digital) written exam including various question types and small Python programming tasks

Duration of the Module

The following number of semesters is estimated for taking and completing the module:
1 Semester.

This module may be commenced in the following semesters:
Wintersemester.

Maximum Number of Participants

This module is not limited to a number of students.

Recap: Lecture 01

Least Squares Linear Regression

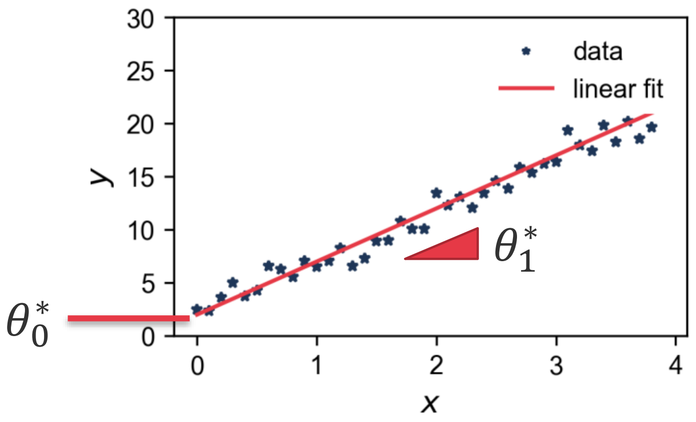
- Scalar variable x and scalar target value y → find $f(x) = \theta_0 + \theta_1 x$

- **Generating process** $y_i = \theta_0 + \theta_1 x_i + \epsilon_i$ with $\mathbf{x} = [1, x_i]$ $y_i = \mathbf{x}_i^\top \theta + \epsilon_i$ $i = 1, \dots, N$
(unobserved random variable ϵ causing deviations from a perfectly linear relationship)

- **Prediction:** $\hat{y} = f(x)$
- Sum of **squared errors**: $\mathcal{L}_{\text{SSE}} = \sum_i (y_i - \hat{y}_i)^2$,$i = 1, \dots N$

- **Solution** for θ_0 and θ_1: vanishing gradient of $\mathcal{L}_{\text{SSE}}$

$$\frac{\partial \mathcal{L}_{\text{SSE}}}{\partial \theta_0} = 0 \text{ and } \frac{\partial \mathcal{L}_{\text{SSE}}}{\partial \theta_1} = 0$$



Recap: Lecture 01

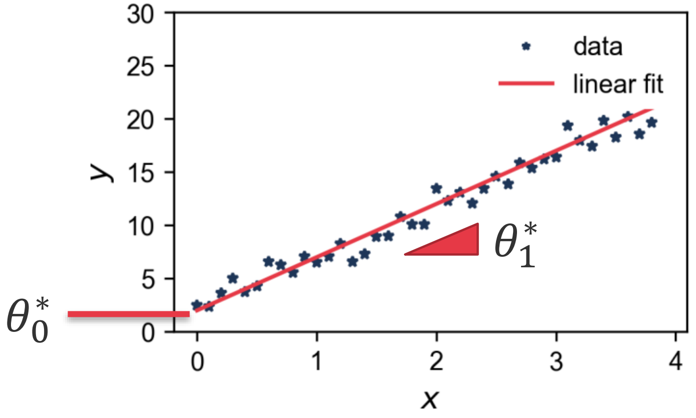
- Loss for scalar setting: $\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - \theta_0 - \theta_1 x_i)^2$

- Vanishing gradient of $\mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = -2 \sum_i (y_i - \theta_0 - \theta_1 x_i) = 0$$
$$\frac{\partial \mathcal{L}}{\partial \theta_1} = -2 \sum_i (y_i - \theta_0 - \theta_1 x_i)\, x_i = 0$$

- Normal equation

$$\underbrace{\begin{bmatrix} N & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \sum_i y_i \\ \sum_i y_i x_i \end{bmatrix}}_{\mathbf{Ax} = \mathbf{b}}$$



→ Solve for $\boldsymbol{\theta}$ to find optimal parameters $\boldsymbol{\theta}^*$

Recap: Lecture 01

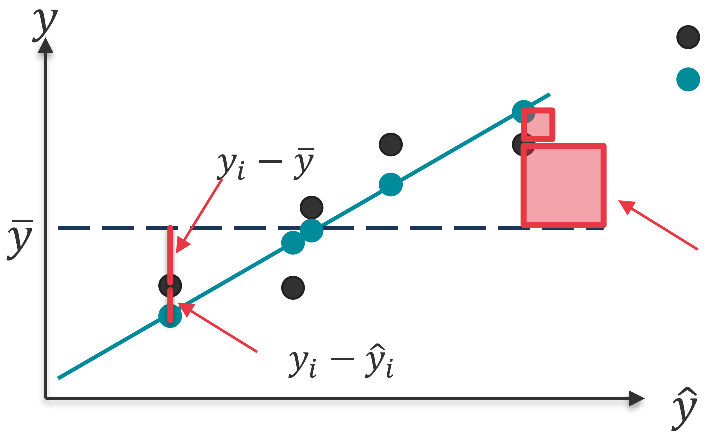
- Measuring the goodness of fit for regression problems: coefficient of determination (R^2 value)

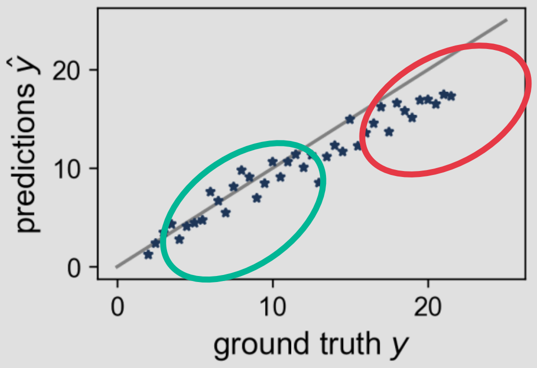
$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \quad i = 1, \dots, N,$$

sample mean $\bar{y} = \frac{1}{N} \sum_i y_i$

Residual sum of squares $\sum_i (y_i - \hat{y}_i)^2$
Total sum of of squars $\sum_i (y_i - \bar{y})^2$

Perfect fit: $R^2 = 1.0$
Baseline model $f(x) = \bar{y}$ $R^2 = 0.0$
‚Worse models' $R^2 < 0.0$





Display of regression model performance

Recap: Exercise 01

Model parameter estimation using Least Squares Linear Regression

- Implementation of the normal form
- Solution using np.linalg.solve
- Return of the coefficients

```
def lin_regress(x: np.ndarray, y: np.ndarray) -> tuple(float, float):
    # assuming x and y being 1-dim np.arrays of
    # length N (number of training data samples)

    # number of training samples
    N = x.shape[0]

    # normal form: A=[N, sum(x); sum(x), sum(x^2)]; b=[sum(y); sum(y*x)]
    A = np.array([[N, np.sum(x)], [np.sum(x), np.sum(x ** 2)]])
    b = np.expand_dims(np.array([np.sum(y), np.sum(y * x)]), axis=-1)


    # solve Ax = b
    theta = np.linalg.solve(A, b).flatten()



    return (theta[0], theta[1])
```

Recap: Exercise 01

Model parameter estimation using
Least Squares Linear Regression

- $P_{\text{engine}} = v(F_{\text{wind}} + F_{\text{roll}})$
- $P_{\text{wo\_wind}} = P_{\text{engine}} - vF_{\text{wind}}$,
  assuming $\alpha = 0, v_{\text{rel}} = v$
- $P_{\text{wo\_wind}} = c_r \cdot mg \cdot v$

takes the form of $y = \theta_0 + \theta_1 x$

```
linear regression model:    y = 1191.6353 + 0.0177 * x
rolling resistance          cR = 0.0177
```

```
data = np.genfromtxt("driving_data.csv", delimiter=",")
velocity = data[:, 0] # m/s
power = data[:, 1] # W

# some constants as given in the exercise sheet
CW, A, RHO, G, M = 0.4, 1.5, 1.2, 9.81, 2400

def wind_resistance(v:np.ndarray)-> np.ndarray:
     return CW * A * (RHO * v ** 2) / 2

power_wo_wind = power - velocity * wind_resistance(v=velocity)
f_roll = power_wo_wind / velocity

y = np.expand_dims(power_wo_wind, axis=1)
X = np.expand_dims(velocity * M * G, axis=1)

# Least-squares lin. regression
theta = lin_regress(x=X, y=y)
theta_0, theta_1 = theta[0], theta[1]

print(f'\nlinear regression model: \ty = {theta_0:.4f} +
{theta_1:.4f} * x')
print(f'rolling resistance \t\t\tcR = {theta_1:.4f}')
```

Agenda

- Attribute types
- Type conversion and encoding
- Python: object-oriented programming

Learning outcomes

Learn to …

- Characterize different attributes by their type

Know about …

- Computational operations valid for specific attribute types
- Variance inflation and the dummy variable trap

Data Types

Data Types

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Most common data types of relevance for this class in Python

data type	`Python` type	description	examples
integer	`int`	whole numbers	`-1`, `0`, `42`
floating point number (real)	`float`	fractional numbers	`-3.14`, `0.0`, `10.1`
string	`str`	sequence of characters	`Hello world!`
boolean	`bool`	logical: true or false	`True`, `False`

- Consider type-hinting when implementing functions!

Attributes

Definition: An attribute is a property or characteristic of an object that may vary, eigher from one dataobject to another or from one time to another.
[Tan, Introduction to Data Mining]

- Example: data object `weather conditions`

- Attributes
  - Current temperature [°C]
  - Current wind speed [m/s]
  - Current wind direction {N, W, S, E}
  - Current humidity [%]
  - Current rain fall {yes, no}
  - Current location [city name]

Types of Attributes

- Attributes can have different **types**
- **Understanding types is crucial for data science and machine learning**
  - The attribute type defines the set of valid operations

- Operations:
  - Distinctness ($=$ and $\neq$)
  - Order ($<$, $\leq$, $>$, $\geq$)
  - Addition and multiplication ($+$, $-$, $*$, $/$)

- Four types of attributes:
  1. **Nominal**
  2. **Ordinal**
  3. **Interval**
  4. **Ratio**

Nominal data

- Observation from **a set of mutually exclusive values, classes or categories**
- Can be expressed in words or in numbers
- There is **no meaningful order** of the labels
- **Arithmetic operations cannot be performed** on nominal data

- Examples:
  - City names
  - Student identity number
  - Political preferences
  - Binary variables (true / false)

Ordinal data

- Observation from **a set values, classes or categories that have a natural rank order**
- Can be expressed in words or in numbers
- There is **a meaningful order** of the labels
- **Arithmetic operations cannot be performed** on ordinal data, but **sorting can be performed**
- **Distances** between categories can be **uneven or undefined**

- Examples:
  - Frequencies {never, rarely, sometimes, often}
  - Colors from a specific color palette
  - University grades {1.0, 1.3, …}
  - Skill levels {beginner, experienced user, expert}

Interval data

- Observation measured on a **numerical interval scale** with …
- **… equal distances between adjacent values**
- **No true zero**, i.e. value=0 is arbitrary and does not indicate the absence of a variable
- There is **a meaningful order** of the labels
- **Arithmetic operations (+ , -) can be performed**

- Examples:
  - Temperatures in [°C] and [F]
  - Time on a 12-hour clock
  - IQ test scores

Ratio data

- Observation measured on a **numerical interval scale** with …
- **… equal distances between adjacent values**
- **There is a true zero**, i.e. value=0 indicates the absence of a variable
- There is **a meaningful order** of the labels
- **Arithmetic operations (+ , -, /, *) can be performed**

- Examples:
  - Temperatures in [K]
  - Speed, height, mass
  - Age

Types of Attributes: Permissible Operations

	Type	Description	Example	Operations
categorical / qualitative	**Nominal**	value corresponds to a set of mutually exclusive values, classes or categories	eye color, city name, brand name, class name, booleans	$=$ and $\neq$
	Ordinal	values from a set of distinct values, can be put into an order	house numbers, study semester, grades	$=$ and $\neq$ $(<, \leq, >, \geq)$
numeric / quantitative	**Interval**	values from a continuous set of equally spaced values, unit of measurement exists, no true zero	temperature °C, time on 12-hour clock, IQ test results	$=$ and $\neq$ $(<, \leq, >, \geq)$ $(+, -)$
	Ratio	values from a continuous set of equally spaced values, unit of measurement exists, true zero indicates absence	age, velocity, height	$=$ and $\neq$ $(<, \leq, >, \geq)$ $(+, -, *, /)$

Types of Attributes: Levels of Measurement

- **Levels of measurement**:
  - metric for precision of data recording and how one can analyze the data
  - the higher, the more complex the recording, and the more options for analysis

	Nominal	Ordinal	Interval	Ratio
Categories	yes	yes	yes	yes
Rank order		yes	yes	yes
Equal spacing			yes	yes
True zero				yes

Types of Attributes

- Example: data object `weather conditions`

Exercise: fill in types

- Attributes
  - Current temperature [°C] → interval
  - Current wind speed [m/s] → ratio
  - Current wind direction {N, W, S, E} → nominal
  - Current humidity [%] → ratio
  - Current rain fall {yes, no} → nominal
  - Current location [city name] → nominal

Ideas for ordinal?

  - Current quarter of the year → ordinal

Types of Attributes

- Why does it matter so much?
- **Computers cannot work with qualitative data** (nominal, ordinal) **directly**
  - A numeric representation of qualitative data is required
  - Numbers may lead to wrong operations on the originally qualitative data

- Example 1: predicting student‘s final grades based on participation and study hours

$$\begin{bmatrix}\text{participation}\\ \text{study hours}\\ \vdots\end{bmatrix} \rightarrow \textbf{model} \longrightarrow [\text{grade}]$$

Ordinal data as **numeric data**
„1.0“ → 1.0
„1.3“ → 1.3
…

Model prediction: $y = 1.801$ → which grade?

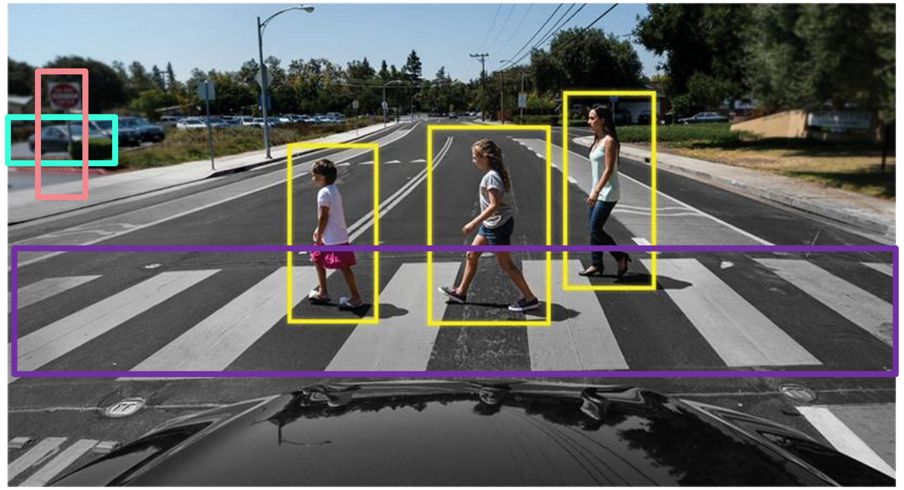
Solution: rounding to nearest valid value → 1.7

Types of Attributes

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Why does it matter so much?
- **Computers cannot work with qualitative data** (nominal, ordinal) **directly**
  - A numerical representation of qualitative data is required
  - Numerics may lead to wrong operations on the originally qualitative data

- Example 2: recognizing traffic participants in a video stream



Nominal data as **numeric data** **(naiive approach)**

Pedestrian → 1
Crosswalk → 2
Stop sign → 3
Car → 4

Model prediction: $y = 1.801$ → interpretation?

Wrong assumption of order and existing distance!

Data Encoding

One-Hot Encoding

- Making qualitative (categorical) data readable to a computer

- **One-Hot Encoding** $y \in \{v_1, \dots, v_k\} \mapsto y \in \mathbb{R}^k, \in \{0, 1\}$, k: number of distinct values / classes

- Traffic example:
  - 4 classes: pedestrian, crosswalk, stop sign, car

  - pedestrian → $[1 \quad 0 \quad 0 \quad 0]^\mathsf{T}$
  - crosswalk → $[0 \quad 1 \quad 0 \quad 0]^\mathsf{T}$
  - stop sign → $[0 \quad 0 \quad 1 \quad 0]^\mathsf{T}$
  - car → $[0 \quad 0 \quad 0 \quad 1]^\mathsf{T}$

  - Model prediction $y = [0.95 \quad 0 \quad 0 \quad 0.05]^\mathsf{T}$ → to 95% a pedestrian, to 5% a car

Do not use for ordinal data, as order gets lost!

(almost) no ML algorithm does create an implicit relationship or order between neighboring values in an output array such as a OHE vector

One-Hot Encoding vs. Multicollinearity

- Example: two-class problem: age over 18 (adult) / age under 18 (child)
- Classical OHE: adult → $[1 \quad 0]$, child → $[0 \quad 1]$

→ perfect collinearity

Solution:

- Select a reference variable, and create a $(K-1)$-dim vector for K categorical variables
- Reference variable maps to vector of zeros (attention with NN output layer activation `softmax`)
- Remaining variables are one-hot encoded (all zeros except for one entry)

$$y \in \{v_1, \dots, v_k\} \mapsto \tilde{y} \in \mathbb{R}^{K-1}, \in \{0, 1\},$$ K: number of distinct values / classes

Multicollinearity (dummy variable trap)

- One feature dimension can be approximated from ≥ 1 other feature dimension(s) using a linear model
  → **feature columns are linearly dependent**
- Large variations in regression model coefficients for small changes to the data set
  → **variance inflation**
- No direct effect on model quality, but
  - → **poor interpretability** of importance of individual feature attributes

- Example: $\tilde{y} = f(x, \theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
  - if $x_1 \approx \alpha_0 + \alpha_1 x_2$, then different β build models of equal quality, e.g. for slightly different X_{train}
  - $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2]$ does not have full rank
  - Check variance inflation factor (VIF)!

Integer (Label) Encoding for Ordinal data

- Encoding ordinal data requires **keeping an order**
- Simplistic encoding: assign an integer to each category, start with 0 for the first category

- **Example**: satisfaction rating for this class
  - 5 classes with natural rank order (*“extremely dislike”, “dislike”, “neutral”, “like”, “extremely like”)*
  - Integer encoding:

extremely dislike	→	0
dislike	→	1
neutral	→	2
…		

- **Caution!** Integer encoding keeps the order, but pretends a measure of (equal) distances!
  - Strictly speaking, a model prediction $\tilde{y} = 1.8$ is not meaningful, and rounding may be wrong
  - Decoding strategy is highly case-specific!

Python: object-oriented programming

Object-oriented programming

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Main reason for using classes and object-oriented programming in the context of ML:

Uniting the location of methods (functionalities) and attributes (data)

- Therefore, class instances have
  - **Methods** (functions):
    perform actions on attributes and external inputs `self.my_attribute(self, x):`
  - **Attributes** (variables):
    assign values to `self.my_attribute`
- Example Class `EmailServer`:
  - Methods: `fetch_new_mails(); check_for_spam(); …`
  - Attributes: `sent_mails; free_space; num_of_mails; …`

Object-oriented programming

- Definition of a class
- Constructor (initialization)
- Attributes (variables)
- Methods (functions)
- Hiding class attributes from user
- Class instantiation

```
class MyClass:

    def __init__(self, some_value='hello‘):
        self.my_attribute = some_value
        self._hidden_attribute = 42

    def a_method(self, some_argument='world‘):
        print(f'{self.my_attribute},{some_argument}‘)

    def _hidden_method(self):
        print(f‘some hidden function‘)


if __name__ == "__main__":

    class_instance = MyClass()
    class_instance.a_method()
    print(class_instance.my_attribute)
```

OOP: linear regression example

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

```python
def fit(x: np.ndarray, y: np.ndarray) -> tuple[float, float]:

      N = x.shape[0] # number of training samples

      # normal form # A = [N, sum(x); sum(x), sum(x^2)]
      # b = [sum(y); sum(y*x)]
      A = np.array([[N, np.sum(x)], [np.sum(x), np.sum(x ** 2)]])
      b = np.expand_dims(np.array([np.sum(y), np.sum(y * x)]),
                         axis=-1)

      # solve Ax = b
      theta = np.linalg.solve(A, b).flatten()
      return (theta[0], theta[1])


def predict(theta: tuple, x: np.ndarray) -> np.ndarray:
      # expects model parameters (theta_0, theta_1) and query
      points x

      # evaluate model at query points
      return theta[0] + theta[1] * x


# fit model and make a prediction
theta = fit(x=x, y=y)
y_hat = predict(theta=theta, x=x_pred)
```

```python
class LinRegressor:
      def __init__(self):
            self.theta: tuple
            self.x_train: np.ndarray
            self.y_train: np.ndarray
            self.N_train: int

      def fit(self, x, y):
            self.x_train = x
            self.y_train = y
            self.N_train = self.x_train.shape[0]

            # normal form: A = [N, sum(x); sum(x), sum(x^2)];
            # b = [sum(y); sum(y*x)]
            A = np.array([[self.N_train, np.sum(self.x_train)],
            [np.sum(self.x_train), np.sum(self.x_train ** 2)]])
            b = np.expand_dims(np.array([np.sum(self.y_train),
            np.sum(self.y_train * self.x_train)]), axis=-1)

            # solve Ax = b
            self.theta = np.linalg.solve(A, b).flatten()

      def predict(self, x):
            return self.theta[0] + self.theta[1] * x


# fit and evaluate lin. regress. model using OOP
regressor = LinRegressor()
regressor.fit(x=x, y=y)
y_hat = regressor.predict(x=x_pred)
```

Exercise 02

November 1st, 2023

Exercise 02: One-Hot Encoding

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Compare functional and object-oriented programming
- Example: One-Hot Encoding categorical data into numeric representation
- Functions / methods:
  - Fit (on some training data)
  - Encode ordinal data
  - Decode numerical data
- Test data: bearing failure modes

	A	B	C
1	true brinelling		
2	excessive load		
3	contamination		
4	loose fits		
5	loose fits		
6	normal fatigue		
7	normal fatigue		
8	excessive load		
9	normal fatigue		
10	false brinelling		
11	misalignment		
12	false brinelling		
13	lubricant failure		
14	excessive load		
15	overheating		

EXCESSIVE LOAD Premature spalled area in ball path. 4	**OVERHEATING** Discoloration of rings, balls and cages. 5	**FALSE BRINELLING** Elliptical wear marks at each ball position. 6
TRUE BRINELLING Ball indentations in raceways. 7	**NORMAL FATIGUE FAILURE** Spalling or flaking of metal from contact surface. 8	**REVERSE LOADING** Balls show grooved wear band. 9
CONTAMINATION Denting of bearing raceways and balls. 10	**LUBRICANT FAILURE** Discolored (blue/brown) ball tracks and balls. 11	**CORROSION** Chemical attack results in reddish/brown discoloration. 12
MISALIGNMENT Raceway ball track not parallel to raceway edges. 13	**LOOSE FITS** Circumferential wear and/or discoloration of mounting surfaces. 14	**TIGHT FITS** Heavy ball wear path at bottom of raceways. 15

from: Schaeffler

Questions?