1 Preparation

1.1 Software installations

Install the required software on a workstation of your choice. You need for the lecture:

1. A git client, e.g. turtoise git under Windows or git under Linux.
2. Python 3.11 or higher
3. A Latex installation, e.g. miktex and texworks under Windows or a corresponding distribution under Linux
4. An installation of Jupyter Lab

1.2 Create and Register Repository (5 points)

Check out the exercise file `Exercise_01.zip` from the canvas course. During the semester, the assignment sheets as well as the associated data and sample solutions will be checked into the course.

Create a publicly accessible repository for your solutions. Name it as follows: `advanced_computer_vision_ss2024_<lastname>_<firstname>`, e.g. `advanced_computer_vision_ss2024_Dahmen_Tim`. In this repository you push your solutions to the tasks. You can branch, push and merge as often as you want, the last commit of the main branch is evaluated.

Please work on your tasks by entering :

1. Unzip the exercise file to the repository. Add the files to version control.
2. Add the corresponding answers to the latex document with the questions. Commit and push both the latex document and a compiled pdf to your solution repository. Please do not rename the task file. However, you can create additional files and link or otherwise use them to structure your solution.
3. Insert your name in the first two lines of the Latex document for each task sheet.
4. Edit the programming tasks by adding the existing source code files. Please do not rename them. Again, you can add your own files or subdirectories and include these new files to structure your solution.

Send an email to Tim.Dahmen@hochschule-aalen.de and let me know that you have created your repository.

I needed the following time to complete the task: 30min

2 Regression, Underfitting, Overfitting

Now edit the Jupyter Notebook `Linear_Regression.ipynb`.

2.1 Data Loader (10 points)

To start, let's look at the function `load_csv_in_sklearn_format` with the following specification:

- The function accepts the file name of a .csv file as an argument.
- The function parses the file and expects two columns (x and y), separated by semicolons, no header line. Example data is the file `linear_dataset_100.csv` or `non_linear_dataset_100.csv`. If something is wrong with the file format, the function throws a FileFormatException.

- The file returns a tuple (x,y). Here x is a list that contains a list with exactly one entry for each line of the file. The entry is the value of the first column as data type float. y is a simple list that contains the value of the second column for each line of the file, again as float. We choose the strange format with the nested list for x because sklearn needs this format later.

a) Argue whether unit tests are needed for this function. If so, which cases should be intercepted and how?
Unit tests are sensible since the data could be provided by a 3rd party and thus we can never be sure if the data, format, etc. is correct. We can test for:

- Is there always one (x, y) pair per row
- Are the values numbers
- Are the values floats

b) Add required unit tests.

c) Implement the function.

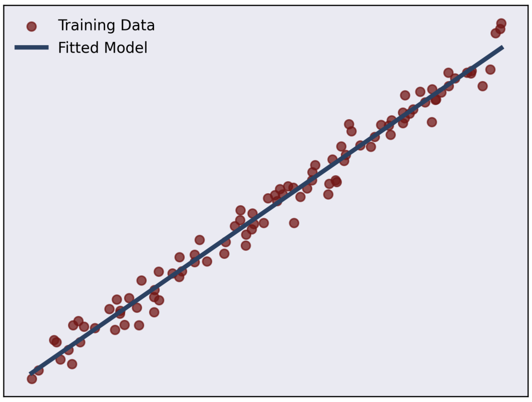
2.2 Linear Regression: Fit (10 points)

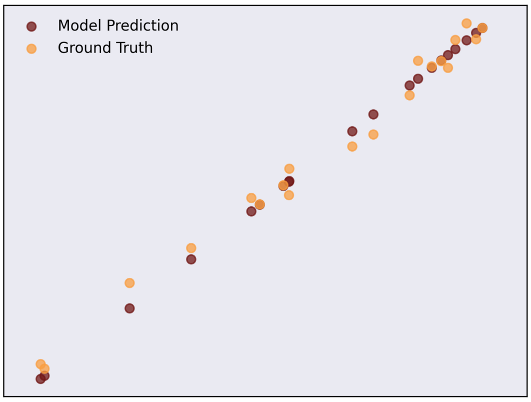
Now load the file `linear_dataset_100.csv`. Implement a linear regression model that is trained on this data. You can use the class linear_model.LinearRegression from sklearn. Now test the model on the test data `linear_dataset_test.csv`.

a) Argue whether unit tests are required for this function. If yes: how could these be designed? If no: how could we make sure that the code works correctly?
In this case I would trust that the scikit-learn authors implemented unit tests for their modules. But we would just fit a model on two points so that we can calculate the parameters of the linear equation ourselves.

b) Is the fit perfect or not? Why is that?
Looking at the fit it seems to be good but not perfect since the ground truth is not directly matching the predictions. Still it is close.

c) Now calculate the r_2 score to determine the fit quantitatively. Argue the advantages and disadvantages of determining the fit optically or numerically.
The qualitative assessment is a good first step to see if the model does what we want. But especially on harder tasks (e.g. egmentation) it might be impossible to tell a difference without quantitative assessment.





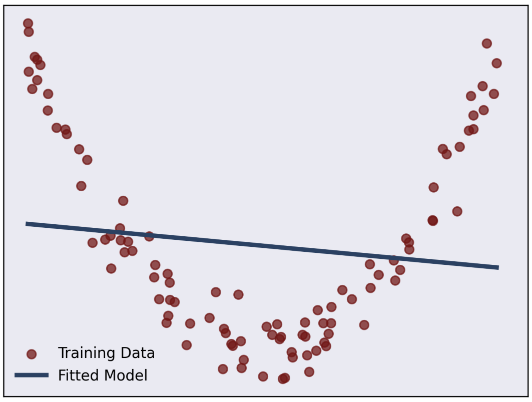
I needed the following time to complete the task: 5min

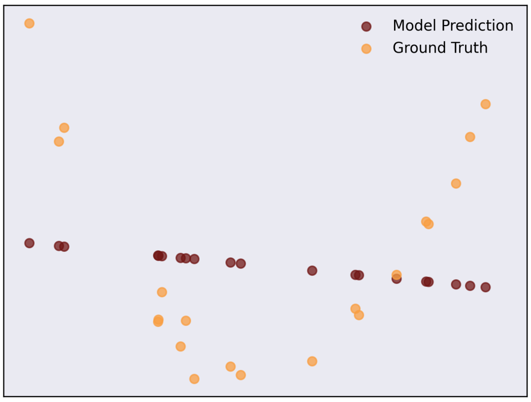
2.3 Linear Regression: Underfit (5 points)

Now repeat the process (load the file, fit, visualize, calculate r2_score) with the files `non_linear_dataset_100.csv` and `non_linear_dataset_test.csv`.

a) Is the fit good or not? Why is that?
No, because a linear model cannot predict data from a 2nd degree polynom.

b) Now calculate the r_2 score to determine the fit quantitatively. Argue how you could determine a cut-off value to determine whether a linear model is sufficient or not.





Recompile the latex to display the results.

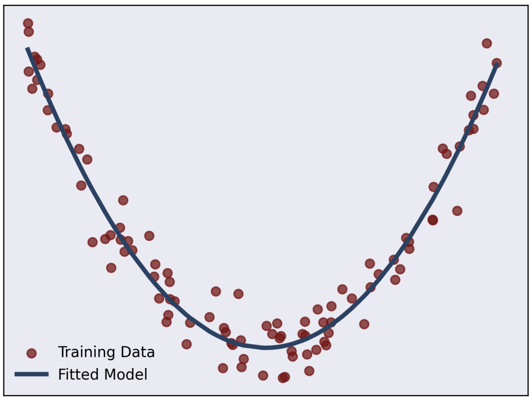
I needed the following time to complete the task: 5min

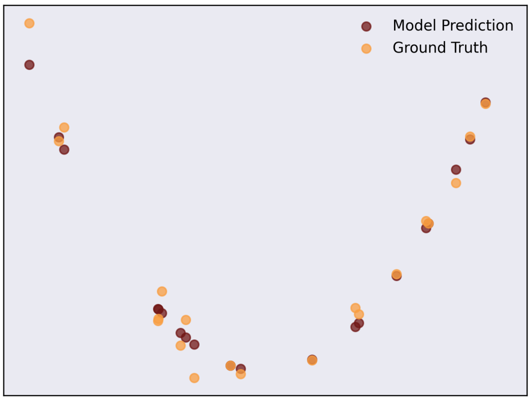
2.4 Polynomial Regression: Fit (10 points)

Now we use a polynomial model. This means that we increase our data by polynomial features (here: 2nd degree) and then carry out a linear regression on the polynomial features. To do this, repeat the process (load the file, fit, visualize, calculate r2_score) with the files `non_linear_dataset_100.csv` and `non_linear_dataset_test.csv`. Use the PolynomialFeatures class from sklearn.preprocessing to create polynomial features.

a) Is the fit good or not? Why?
Yes, because a polynomial regression model can fit data from a polynom.

b) Now calculate the r_2 score to determine the fit quantitatively. Argue how you could determine a cut-off value to determine whether a linear model is sufficient or not.





Recompile the latex to display the results.

I needed the following time to complete the task: 5min

2.5 Piecewise Polynomial Regression: Overfit (10 points)

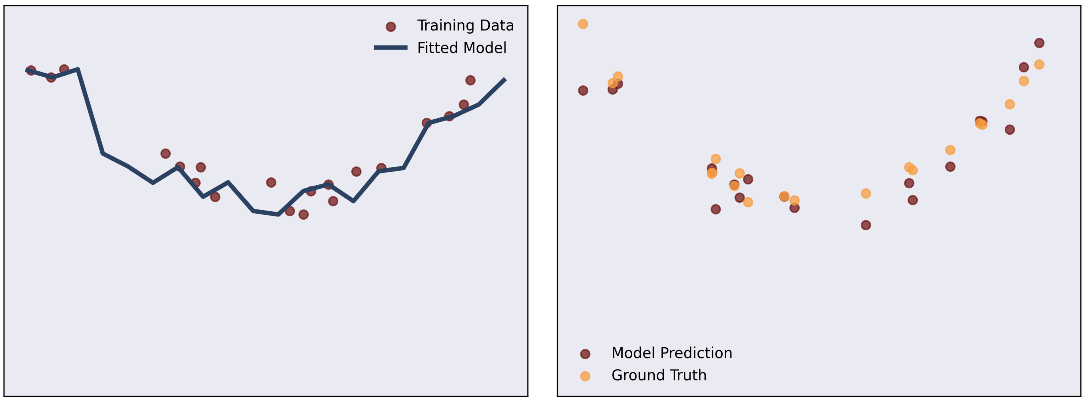
Now we use a spline model (piecewise polynomial). To do this, we construct a spline object (degree 2) using the sklearn function make_interp_spline and use this to predict the test data.

a) Implement the regression.

b) Is the fit good or not? Why?
Not really, because the model is overfitting (Learning the training data points).

c) Now calculate the r_2 score to determine the fit quantitatively.

d) How does the fit change if you use splines of higher degree (3 or 5)? Does the result get better or worse? Why?
Worse because the model learns to fit the noise of the training data, which is obviously different in the test data.



Recompile the latex to display the results.

I needed the following time to complete the task: 5min