BJ to Lab-1

Lab-1.

The term Machine Learning (ML) refers to the automated detection of meaningful patterns in data.

0. Basic Task: Attend lectures and know/understand ML theory.

Podstawowe zadanie: Uczestniczyć w wykładach i znać teorię ML, w tym rozumieć podstawowe pojęcia, takie jak

Types of learning: Supervised versus Unsupervised, Active versus Passive Learners,…

ERM – learning (hypothesis) [Empirical Risk Minimization]
PAC – learning [Probably Approximately Correct (PAC) learning] PAC Learnability
Agnostic PAC (Removing the Realizability Assumptio), Agnostic PAC Learnability
VC-Dimension
The No-Free-Lunch Theorem
Non-uniform Learnability
Structural Risk Minimization
Minimum Description Length and Occam's Razor
The Runtime of Learning
Etc…

From Theory to Algorithms

Głównym celem Laboratorium jest:
From theory and algorithms to practical solutions of the specific problems
(using the Python language and the Orange system)

Lab-1: Predyktory liniowe:

1.1 Klasyfikacja binarna

1.2 Regresje liniowe [i nieliniowe/wielomianowe].

Lab.: From Algorithms to coding
Lab-1: Linear predictors:
Task 1.1 Binary Classification
Task 1.2 Linear [and nonlinear] regressions

We start from:

Task 1.01: Generation data for subsequent use and storing
Task 1.02: Reading data from files

Goal:
Lab-1: Creating some matrices with data and using them within Binary Classification and lin/non-lin Regression.

Storing data

v1_basic:
| Feature 1 | Feature 2 | ... | Feature n | **Target (Label)** |

v2_*possible*
| **Target (Label)** | Feature 1 | Feature 2 | **...** | Feature n |

1. Many machine learning libraries have parameters or methods that allow you to explicitly specify where the label vector is located in the data. This can be especially useful when the data structure varies, and you need to inform the library where to find the labels. Therefore, if you have the option to specify in advance which column contains the labels, you can avoid confusion.
2. However, to ensure that your code interacts correctly with a specific library, it's recommended to read the documentation of that library. In some cases, there might be a standard expectation that labels are located in a particular column (e.g., the last one), but this can vary depending on the library. Clarifying the library's requirements in the documentation will help you correctly organize your data for working with it.

1. `X_last_col = data_last_col[:, :-1]`: Here, `data_last_col[:, :-1]` selects all rows (`:`) and all columns except the last one (`:-1`). It is essentially extracting the feature matrix `X_last_col` from `data_last_col` by excluding the last column.
2. `y_last_col = data_last_col[:, -1]`: Here, `data_last_col[:, -1]` selects all rows (`:`) but only the last column (`-1`). It extracts the column containing labels and assigns it to the variable `y_last_col`. This assumes that the labels are stored in the last column of the matrix.

So, in the context of reading a matrix with labels in the last column, the usage of -1 helps in indexing the last element along a particular axis (in this case, the last column). It's a convenient way to reference the last element without explicitly specifying its position, especially when the number of columns is not known in advance.

Examples:

```
import numpy as np

# Set the random seed for reproducibility
np.random.seed(42)

# Parameters
m1 = 15
m = 2 * m1
n = 3

# 1. Label in the 1st column
filename_first_col = 'D:/Python_files_data/ml_matrices/gen_mat_10_1stcol_csv_30.csv'

# Generate and save the matrix with label in the 1st column
with open(filename_first_col, 'w') as file:
a1, b1 = 2, 5
a2, b2 =6, 10
    for _ in range(m1):
        # Class label -1
        row = ['-1']
        # Generate random values for the features
        row += [f'{np.random.uniform(a1, b1):.2f}' for _ in range(n)]
        file.write(','.join(row) + '\n')

        # Class label 1
        row = ['1']
        # Generate random values for the features
        row += [f'{np.random.uniform(a2, b2):.2f}' for _ in range(n)]
        file.write(','.join(row) + '\n')

# Print result for label in the 1st column
print(f"Matrix with label in the 1st column created and saved to {filename_first_col}")

# 2. Label in the last column
# Path for the 2nd variant of forming matrix
filename_last_col = 'D:/Python_files_data/ml_matrices/gen_mat_11_lastcol_csv_30.csv'

# Generate and save the matrix with label in the last column
with open(filename_last_col, 'w') as file:
    for _ in range(m1):
        # Generate random values for the features
        row = [f'{np.random.uniform(a1, b1):.2f}' for _ in range(n)]
        # Class label -1
        row.append('-1')
        file.write(','.join(row) + '\n')

        # Generate random values for the features
        row = [f'{np.random.uniform(a2, b2):.2f}' for _ in range(n)]
        # Class label 1
        row.append('1')
        file.write(','.join(row) + '\n')
```

```
# Print result for label in the last column
print(f"Matrix with label in the last column created and saved to {filename_last_col}")

# 3. Read the matrices formed
# Reading matrix with label in the 1st column
data_first_col = np.genfromtxt(filename_first_col, delimiter=',')
X_first_col = data_first_col[:, 1:]
y_first_col = data_first_col[:, 0]

# Reading matrix with label in the last column
data_last_col = np.genfromtxt(filename_last_col, delimiter=',')
X_last_col = data_last_col[:, :-1]
y_last_col = data_last_col[:, -1]
```

About normalization of input data/matrix within ML problems:

1. **Input Data Normalization in ML:**
   - In many machine learning algorithms, especially those that rely on distance metrics or gradient-based optimization, normalizing input data can be crucial.
   - Normalization helps ensure that features with larger scales do not dominate the learning process.
2. **Binary Classification with Linear Programming (LP):**
   - Linear Programming typically deals with optimization problems rather than classification. If you are referring to linear classifiers, such as Support Vector Machines (SVM) with linear kernels, normalization is often recommended for better performance.
3. **Binary Classification with Logistic Regression:**
   - Logistic Regression can benefit from input normalization. It's common to apply normalization to ensure that the logistic regression model converges faster during training.
4. **Linear Regression (Labels - Real Numbers):**
   - Normalization in linear regression depends on the specific algorithm and optimization method. It might not be strictly compulsory, but normalizing can aid the optimization process.
5. **Non-Linear Regression:**
   - Similar to linear regression, the necessity of normalization in non-linear regression depends on the underlying

Summary, while normalization is often recommended for various machine learning tasks, its strict necessity depends on the specific algorithms and optimization methods you are using. It's good practice to experiment with both normalized and non-normalized data and observe the impact on model performance.

The choice of normalization method can indeed impact the output results and subsequent decision making in ML. Different normalization methods might be more suitable for certain types of data or specific algorithms. Here are a few points to consider:

1. **Impact on Distance Metrics:**
   - If your machine learning algorithm relies on distance metrics (e.g., k-Nearest Neighbors), the choice of normalization method can affect the calculation of distances between data points.
2. **Effect on Optimization Convergence:**
   - In optimization-based algorithms (e.g., gradient descent), normalization can influence the convergence speed and stability of the optimization process. Some algorithms may converge faster with normalized data.
3. **Scale Sensitivity of Algorithms:**
   - Some algorithms are sensitive to the scale of input features. For example, Support Vector Machines (SVM) can be sensitive to the scale of features, making normalization important.
4. **Interpretability of Coefficients:**
   - In linear models, normalization can impact the interpretability of coefficients. Without normalization, coefficients might represent the change in the target variable for a one-unit change in the corresponding feature, which may not be meaningful if the features are on different scales.
5. **Applicability to Domain Knowledge:**
   - Consider domain knowledge and the nature of the data when choosing a normalization method. Min-Max scaling, z-score normalization, and robust scaling are common techniques, but their suitability may vary based on the data distribution and characteristics.

It's recommended to experiment with different normalization methods and evaluate their impact on model performance. Cross-validation can be used to assess the generalization performance of models with different normalization strategies. Ultimately, the choice should be guided by empirical observations and the specific characteristics of your data and task.

Below are Python code snippets for different normalization approaches using the NumPy library. These functions take a dataset as input and return the normalized dataset. Each function corresponds to a different normalization method.

```
import numpy as np
def min_max_scaling(data):
    min_vals = np.min(data, axis=0)
    max_vals = np.max(data, axis=0)
    normalized_data = (data - min_vals) / (max_vals - min_vals)
    return normalized_data

def zero_one_scaling(data):
    return min_max_scaling(data)

def min_max_scaling_custom_range(data, range_min, range_max):
    min_vals = np.min(data, axis=0)
    max_vals = np.max(data, axis=0)
    normalized_data = range_min + (data - min_vals) * (range_max - range_min) / (max_vals - min_vals)
    return normalized_data
```

```
def z_score_standardization(data):
    mean_vals = np.mean(data, axis=0)
    std_devs = np.std(data, axis=0)
    standardized_data = (data - mean_vals) / std_devs
    return standardized_data

def mean_shift(data):
    mean_vals = np.mean(data, axis=0)
    shifted_data = data - mean_vals
    return shifted_data

# Example usage:
# data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# normalized_data = min_max_scaling(data)
# print(normalized_data)
```
In these functions:

- min_max_scaling: Performs Min-Max scaling, mapping values to the [0, 1] range.
- zero_one_scaling: An alias for min_max_scaling.
- min_max_scaling_custom_range: Allows scaling to a custom range (specified by range_min and range_max).
- z_score_standardization: Performs Z-score standardization.
- mean_shift: Centers the data by subtracting the mean of each feature.

Robust normalization, often referred to as Robust Scaling or Robust Standardization, is a normalization technique that aims to make a dataset resistant to the influence of outliers. It is particularly useful when the dataset contains extreme values that can disproportionately affect other normalization methods. In Robust Normalization, the median and the interquartile range (IQR) are used instead of the mean and standard deviation. The median is a robust measure of central tendency, and the IQR is a robust measure of data dispersion.

```
import numpy as np
def robust_normalization(data):
    median_vals = np.median(data, axis=0)
    q1 = np.percentile(data, 25, axis=0)
    q3 = np.percentile(data, 75, axis=0)
    iqr = q3 - q1
    normalized_data = (data - median_vals) / iqr
    return normalized_data
# Example usage:
# data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [100, 200, 300]])
# normalized_data = robust_normalization(data)
# print(normalized_data)
```
In this function:

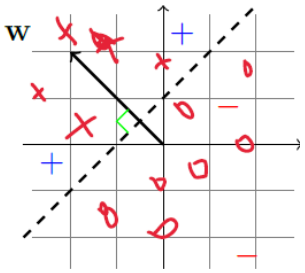
- median_vals: Calculates the median for each feature.
- q1 and q3: Calculate the 25th and 75th percentiles for each feature, respectively.
- iqr: Computes the interquartile range for each feature.
- normalized_data: Normalizes the data using the median and IQR.

Robust Normalization is beneficial when dealing with datasets that exhibit skewness or contain outliers, as it provides a more robust estimate of the central tendency and spread of the data.

Task 1.1 Binary Classification

Klasyfikacja binarna
- chcemy przewidywać klasę, -1 lub 1; (alternatywnie - prawdopodobieństwo klasy 1)
- np. czy ma nowotwór, czy udzielić kredytu, spam czy nie spam
- idea: użyjmy regresji liniowej
- problem: na wyjściu jest liczba rzeczywista

An Example, Fig.



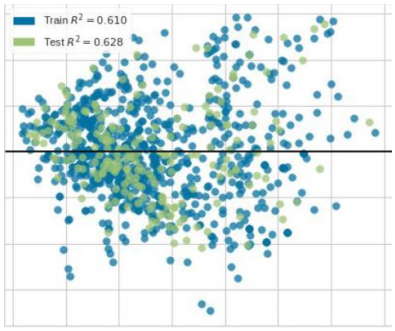
1. LP:

zastosowanie metod programowania liniowego (LP):
Chcemy znaleźć (napisać równanie) hiperpłaszczyznę w przestrzeni X dimX=d, która dzieli dane na 2 (rozłączne) grupy.
Problem został rozwiązany **tylko** dla **przypadku Realizowalnego (*Realizable case*)**

$$\hat{y} = w_0 1 + w_1 x_1 + ... + w_d x_d = x^T w$$

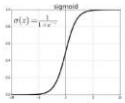
use of Linear Programming (LP) methods: We want to Find (write an equation) a hyper-plane in X space in $\mathbb{R}^d$ that divides the data into 2 (disjoint) groups. The problem is solved **only** for the **Realizable case**

Unrealizable case:



2. Bin. Classification with Logistic Regression (LR)
(Regresja logistyczna)

Model regresji logistycznej

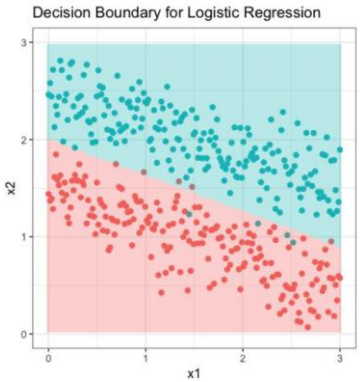


- model to przekształcenie regresji liniowej z użyciem funkcji logistycznej (sigmoidy)

$$\hat{y} = \sigma(Xw) = \frac{1}{1 + \exp(-Xw)} = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + ... + w_d x_d)}}$$

Interpretation:
prawdopodobieństwo klasy pozytywnej

$$\hat{y} = p(x) = P(y = 1|x)$$



Hiperpłaszczyznę można zaimplementować w problemach klasyfikacji binarnej tylko w przypadku Realizowalnym.
Jednocześnie zastosowanie regresji logistycznej do problemów klasyfikacji binarnej jest zawsze możliwe, a przy zastosowaniu kodu standardowego można uzyskać błędy klasyfikacyjne, ich liczbę oraz konkretne punkty w przestrzeni, które są błędnie sklasyfikowane.

Bin. Classification, Libraries

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from scipy.optimize import linprog

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

Task 1.2 Linear [and nonlinear] regressions

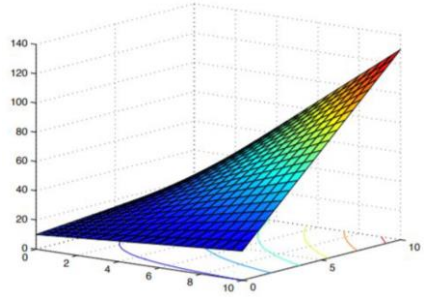
1. Linear Regression (LR):
Regresja liniowa

$$\hat{y} = w_0 1 + w_1 x_1 + ... + w_d x_d = x^T w$$

● założenia:
○ osobny współczynnik dla każdej wejściowej wartości
○ przewidywana wartość to kombinacja liniowa cech i wag
● zastosowanie jako baseline - najprostszy model;

2. Non-Linear Regression (NLR) → LR

$$y = 10 + x_1 + x_2 + x_1 x_2 = w \cdot x$$
$$x = [1, x_1, x_2, x_1 x_2]$$
$$w = [10, 1, 1, 1]$$



Liniowość regresji nieliniowej
● regresja liniowa to kombinacja liniowa cech i wag
● może używać nieliniowych transformacji cech
● inżynieria cech (feature engineering) - tworzenie nowych cech, przekształcanie istniejących, usuwanie mało ważnych itp.
● proces tworzenia nowych cech na podstawie istniejących da się zautomatyzować
○ https://towardsdatascience.com/automated-feature-engineering-in-python-99baf11cc219

Regresja wielomianowa
● idea: zamiast cech używamy ich wielomianu N-tego stopnia:

$$[a, b, c, d] \rightarrow [a, b, c, d, a^2, b^2, c^2, d^2, ab, ac, ad, bc, bd, cd]$$

● używamy bazy wielomianowej jako nieliniowego przekształcenia cech
● można generować też same interakcje (iloczyny)
● cechy wielomianowe zwiększają pojemność modeli
○ duża pojemność - możliwa reprezentacja skomplikowanych zależności, przeuczenie przy małej liczbie przykładów

Theory (see Lectures!)

Obliczanie współczynników w

- minimalizujemy sumę kwadratów błędów

$$\arg\min_w \sum_{i=1}^{n} (y_i - x_i^T w)^2$$

Rozwiązanie analityczne - równania normalne

- liczymy pochodną, przyrównujemy do zera i dostajemy równania normalne (normal equations):

$$w = (X^T X)^{-1} X^T y$$

- kosztowne obliczeniowo!
- w praktyce można zastosować Singular Value Decomposition (SVD)

Rozwiązanie numeryczne

- przyjmujemy jako funkcję kosztu błąd średniokwadratowy (mean squared error, MSE), podzielony dodatkowo przez 2 dla ułatwienia obliczeń

$$L(w) = \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - x_i^T w\right)^2 = \frac{1}{2n}||y - Xw||^2$$

- trzeba obliczyć gradient, czyli pochodną po elementach w
- mamy unikalne rozwiązanie, bo funkcja kosztu jest wypukła (sprawdzenie, że hesjan jest dodatnio określony)

(Gradient descent)

Basic:

Metryki

- Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

Others:

- Root Mean Squared Error - ta sama jednostka, co dane

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2}$$

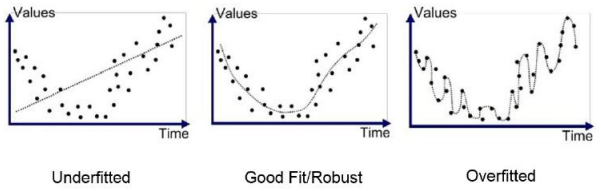
- Mean Absolute Error (MAE)

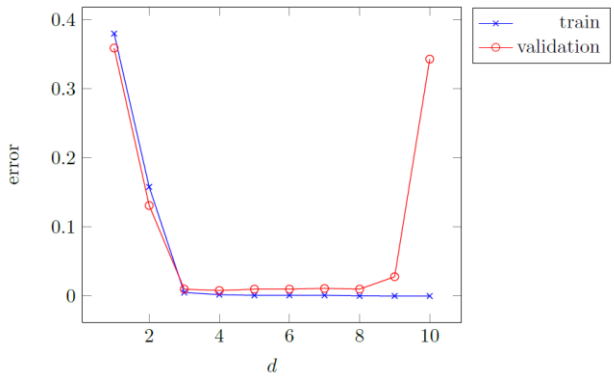
$$\text{MAE} = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_1|}{n}$$

- R^2 - jak dużo wariancji zmiennej y wyjaśnia model

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Model Selection and Validation
Training set: S_t,
Validation (testing) set: S_v.





<u>Approaches to Validation:</u>

1. *Usual*: S= S_t + S_v (t – training, v – validation)

2. *k-Fold Cross Validation*:
In k-fold cross validation the original training set is partitioned into k subsets (folds) of size m/k (for simplicity, assume that k1=m/k is an integer). For each fold, the algorithm is trained on the union of the other folds and then the error of its output is estimated using the fold. Finally, the average of all these errors is the estimate of the true error.

The special case k = m, where m is the number of examples, is called *leave-one-out* (LOO).

3. *Train-Validation-Test* split : S= S_t + S_v + S_t

In most practical applications, we split the available examples into three sets. The 1st set is used for training our algorithm and the second is used as a validation set for model selection. After we select the best model, we test the performance of the output predictor on the *third* set, which is often called the *test set*."
The number obtained is used as an estimator of the true error of the learned predictor.

Laboratorium i zadanie domowe:

1. Symuluj za pomocą generatora liczb losowych, specjalna (przemyślana) matryca danych dla:
Eg: see in the very beginning

1.1 Klasyfikacja binarna: zastosowanie modelu *liniowego lub logistycznego*:
Eg.:

1.2 Rozwiązywanie problemów regresyjnych – Liniowych
1.3 Wielomianowych.
Eg.:

```
import numpy as np
import pandas as pd
from scipy.stats import norm

# Set the seed for reproducibility
np.random.seed(42)

# Set the number of observations
n = 200
a1, b1, a2, b2 = 1, 11, 2, 10

# Generate random numbers x and y within specified intervals
x = np.random.uniform(a1, b1, size=n)
y = np.random.uniform(a2, b2, size=n)

# Calculate the value of the function z1
z1 = (x - 6)**2 + (y - 6)**2 + 1

# Generate a noise: random numbers with normal distribution
# using z1 and 0.1*z1 as mean and standard deviation
z = np.random.normal(loc=z1, scale=0.1*z1, size=n)

# Create a DataFrame with the data
data = pd.DataFrame({'z': z, 'x_1': x, 'x_2': y})
#data[:, 0] += noise

# Save the data to a CSV file
filename = 'D:/Python_files_data/ml_matrices/matr_6_nl_regr_200.csv'
data.round(3).to_csv(filename, index=False)  # Save with 3 decimal places

print(f'Data saved to {filename}')

k = 3
# VARIANTs of PRINTING -))
# Print the first k rows of data
print(f"First {k} rows of the data:\n{data.head(k).round(3)}")

# Print the first 3 elements of the 1st column
print(f"First k = {k} elements of the 1st column:\n{data['z'].head(k).round(3)}")

# Print the first 3 elements of the 1st column
#print(f"First k = {k} elements of the 1st column:\n{data['z'].head(k):.3f}")        #mistakes !-((

# Print the first 3 elements of the 1st column
print(f"another pandas approach: First k = {k} elements of the 1st column:\n{data['z'].head(k).apply(lambda x: f'{x:.3f}')}")
```

2. Zrealizować zadanie klasyfikacji binarnej dowolną metodą, wykorzystując macierz stworzoną do klasyfikacji binarnej.

```
import numpy as np
from sklearn.linear_model import LogisticRegression

# Load the data
file_path = 'D:/Python_files_data/ml_matrices/bin_class_100.csv'
bin_class_100 = np.loadtxt(file_path, delimiter=',')

# Extract labels and features
y = bin_class_100[:, 0]
X = bin_class_100[:, 1:]

# Create a logistic regression model
model = LogisticRegression()

# Fit the model to the data
model.fit(X, y)

# Extract coefficients of the plane equation
w = model.coef_[0]
b = model.intercept_[0]
```

```
print("Equation of the plane:")
print(f"{w[0]} * x1 + {w[1]} * x2 + {w[2]} * x3 = {b}")

# Make predictions
predictions = model.predict(X)

# Calculate the risk (error)
risk = np.mean(predictions != y)
print(f"Risk (Error): {risk}")

# Find misclassified points
misclassified_indices = np.where(predictions != y)[0]

# Print the number of misclassified points
print(f"Number of misclassified points: {len(misclassified_indices)}")

# Print details of misclassified points
print("Details of misclassified points:")
print("Line Number, Label, x1, x2, x3")
for index in misclassified_indices:
    point = bin_class_100[index]
    print(f"{index + 1}, {int(point[0])}, {point[1]:.2f}, {point[2]:.2f}, {point[3]:.2f}")
```

Results:

Equation of the plane:
2.3877551644937207 * x1 + 2.437806280808331 * x2 + 2.583930624462916 * x3 = -3.648188108575438
Risk (Error): 0.06
Number of misclassified points: 6
Details of misclassified points:
Line Number, Label, x1, x2, x3
15, 1, 0.15, 0.65, 0.51
24, -1, 0.83, 0.25, 0.53
39, 1, 0.73, 0.05, 0.43
62, -1, 0.11, 0.68, 0.70
78, -1, 0.56, 0.65, 0.56
82, -1, 0.65, 0.73, 0.28
Process finished with exit code 0

3. Wybierz regresję liniową i wielomianową 2 i 3 stopnia, wykorzystując macierz specjalnie (i przemyślanie!) stworzoną wcześniej do tego zadania.
Aby wybrać najlepszy model, użyj oceny MCE, aby zasadniczo podzielić zbiór danych na podzbiory uczące i walidacyjne.

Zaleca się również stosowanie metod *k-Fold Cross Validation* lub *Train-Validation-Test* split.
Eg:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Load data
filename = 'D:/Python_files_data/ml_matrices/matr_6_nl_regr_200.csv'
data = pd.read_csv(filename)

# Extract features (x1, x2) and target variable (z)
X = data[['x_1', 'x_2']]
y = data['z']

# Split the data into training and verification sets
n_t = 100  # number of training samples
X_train, X_verify, y_train, y_verify = train_test_split(X, y, test_size=n_t, random_state=42)

# Create polynomial features for degree=2
poly2 = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly2 = poly2.fit_transform(X_train)
X_verify_poly2 = poly2.transform(X_verify)

# Fit linear regression model
model2 = LinearRegression()
model2.fit(X_train_poly2, y_train)

# Predictions on training and verification sets
y_train_pred2 = model2.predict(X_train_poly2)
y_verify_pred2 = model2.predict(X_verify_poly2)

# Calculate MSE
mse_train2 = mean_squared_error(y_train, y_train_pred2)
mse_verify2 = mean_squared_error(y_verify, y_verify_pred2)

# Results
```

Model M2 - degree=2:
x1=x, x2=y, x3=xy, xy=x^2, x5=y^2
Coefficients: ['-11.82', '-12.42', '0.97', '0.02', '1.03']
Intercept: 73.88
MSE on Training Set: 2.38
MSE on Verification Set: 3.61
Process finished with exit code 0

Lab-1.

Laboratorium nr 1 zaczniemy od następującego prostego problemu.

Samochód jechał z punktu A do punktu B z prędkością v1=70 km/h, a z B do A z prędkością v2=30 km/h.
4.1 Jaka jest średnia prędkość **v** samochodu na trasie ABA?

4.2 Rozwiązać zadanie dla przypadku, gdy prędkości są zmiennymi losowymi w pobliżu zadanych (rozważ przypadek niezależnych i zależnych zmiennych losowych).

Zadanie domowe (1-3 + 4.2)

++

See to choose a real-world case study:

https://my.nsta.org/search?q=regression&_gl=1*3obztl*_ga*OTE0MTM2OTk1LjE3MDk1NTU2NzM.*_ga_FT5S2XN5CQ*MTcwOTU1NTY3My4xLjEuMTcwOTU1NTcwNS4wLjAuMA