Data Mining & Machine Learning

Computer Exercise 7 - Linear Models

Steinarr Hrafn Höskuldsson

October 2022

Section 1.2

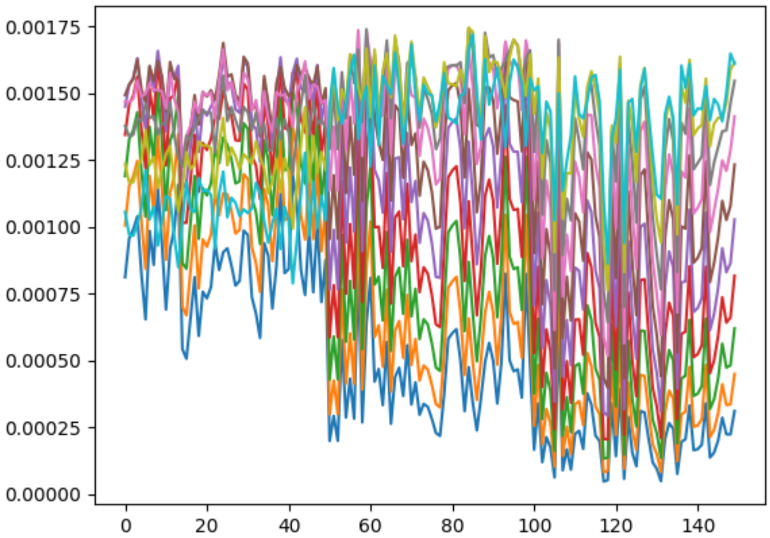

Figure 1: The output of each of the basis function over the dataset

Section 1.5

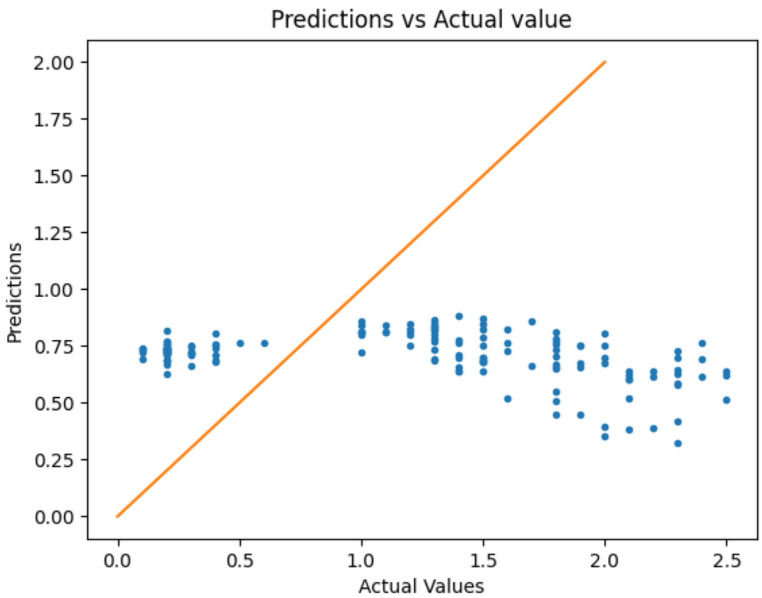

Figure 2: The Actual values plotted in a scatter plot against the predicted values.

Figure 2 shows the Actual values plotted against the predicted values. There is practically no correlation between them.

Independent

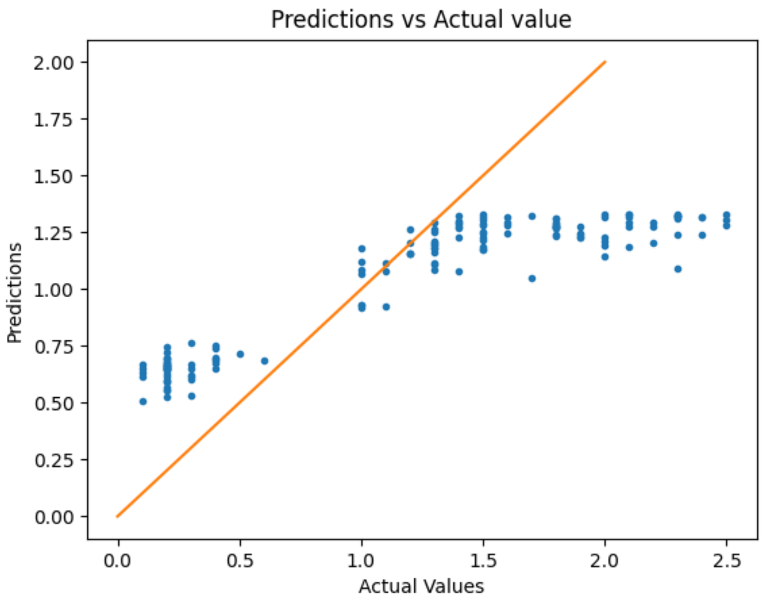

Figure 3: The Actual values plotted in a scatter plot against the predicted values. After fixing the colomn/row mix up

A fellow student, Sigurður Ágúst Jakobsson, pointed out on Piazza that we are mixing up columns and rows when finding the min and max of each feature. By fixing that we get a slightly better scatter plot as can be seen in Figure 3. And a little bit of trial and error with changing `M` and `sigma`, gives the scatter plot in Figure 4.

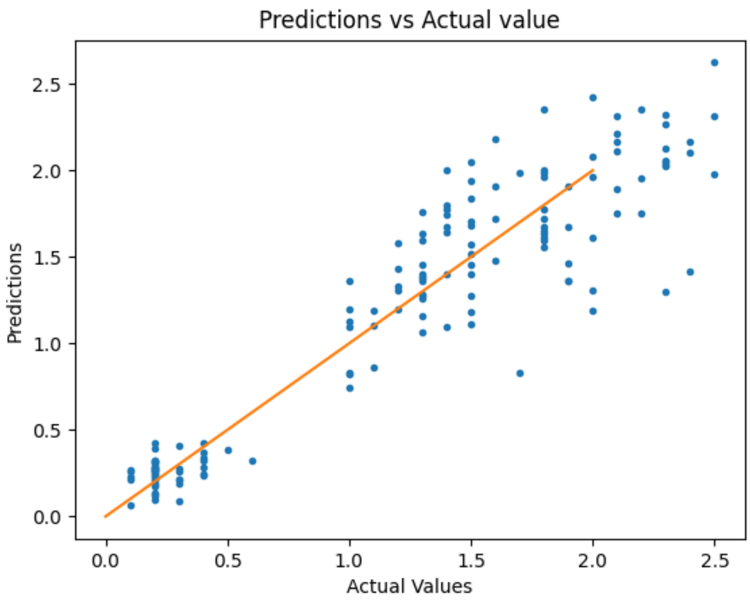

Figure 4: The Actual values plotted in a scatter plot against the predicted values. After choosing M=100 and sigma=2

In an effort to find the optimal values for `M` and `sigma` a for loop was written that goes through different value combinations and saves the Mean-Square-Error of each combination.

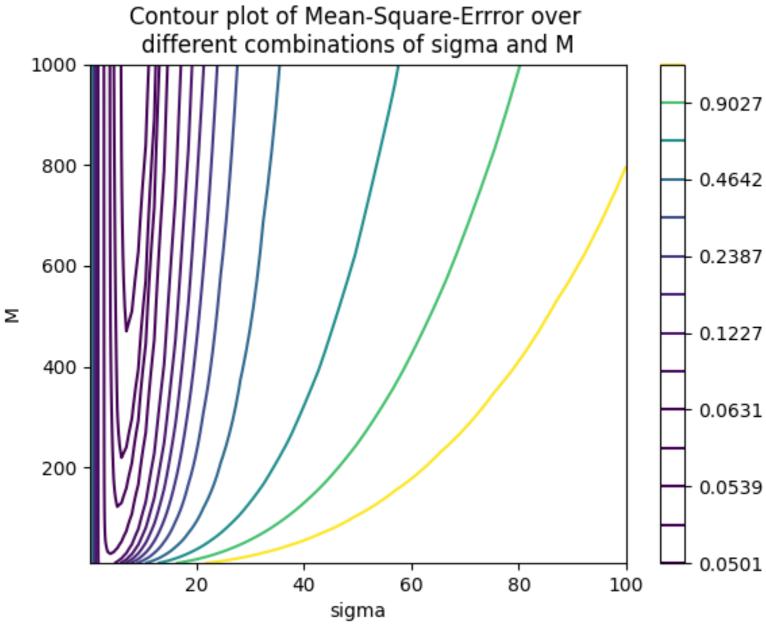

Figure 5: Contour plot of the Mean-Square-Error of the predicted vs actual values with different combinations of M and sigma. Note that the values of the contour lines are not evenly spaced.

The results were plotted in a contour plot, seen in Figure 5. The best performance was with `M` = 1000 and `sigma` = 7.9.

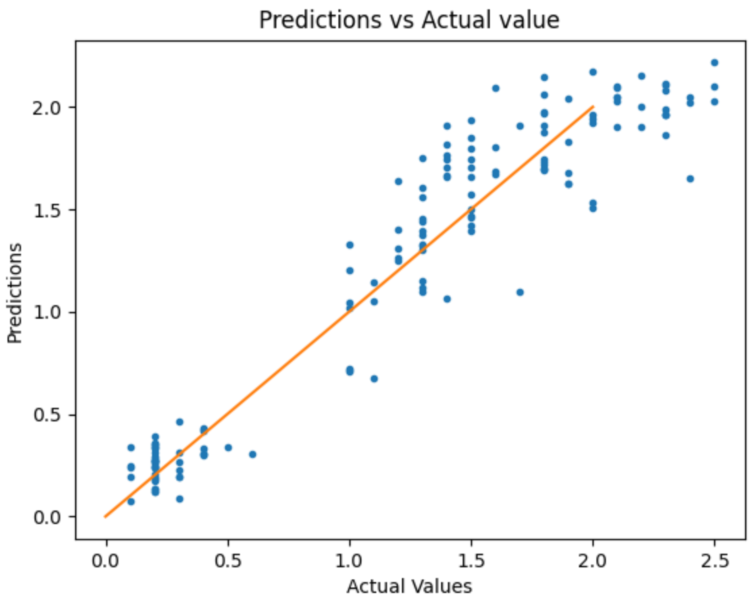

Figure 6: The Actual values plotted in a scatter plot against the predicted values. with M=1000 and sigma=7.9

A scatter plot of Actual vs Predicted values with `M` = 1000 and **`sigma`** = 7.9 was made and can be seen in Figure 6

Appendix

A Code

```python

# Author:
# Date:
# Project:
# Acknowledgements:
#

# NOTE: Your code should NOT contain any main functions or code that is executed
# automatically.  We ONLY want the functions as stated in the README.md.
# Make sure to comment out or remove all unnecessary code before submitting.

import numpy as np
import matplotlib.pyplot as plt

from tools import load_regression_iris
from scipy.stats import multivariate_normal


def mvn_basis(
    features: np.ndarray,
    mu: np.ndarray,
    sigma: float
) -> np.ndarray:
    '''
    Multivariate Normal Basis Function
    The function transforms (possibly many) data vectors <features>
    to an output basis function output <fi>
    Inputs:
    * features: [NxD] is a data matrix with N D-dimensional
    data vectors.
    * mu: [MxD] matrix of M D-dimensional mean vectors defining
    the multivariate normal distributions.
    * sigma: All normal distributions are isotropic with sigma*I covariance
    matrices (where I is the MxM identity matrix)
    Output:
    * fi - [NxM] is the basis function vectors containing a basis function
    output fi for each data vector x in features
    '''
    M = mu.shape[0]
    ret = []
    for i in range(M):
        ret.append(multivariate_normal(mu[i, :], sigma).pdf(features))
    return np.array(ret).T



if __name__ == "__main__":
    X, t = load_regression_iris()
    N, D = X.shape
    M, sigma = 100, 2
    mu = np.zeros((M, D))
    for i in range(D):
        mmin = np.min(X[:, i])
        mmax = np.max(X[:, i])
        mu[:, i] = np.linspace(mmin, mmax, M)
```

```python
    # print(N, D, M)
    # print(mu)
    fi = mvn_basis(X, mu, sigma)
    # print(fi)

def _plot_mvn():
    X, t = load_regression_iris()
    N, D = X.shape
    M, sigma = 10, 10
    mu = np.zeros((M, D))
    for i in range(D):
        mmin = np.min(X[i, :])
        mmax = np.max(X[i, :])
        mu[:, i] = np.linspace(mmin, mmax, M)
    fi = mvn_basis(X, mu, sigma)
    plt.figure("plot_1_2")
    plt.plot(fi)
    plt.savefig("07_linear_models/plot_1_2")
    # plt.show()

if __name__ == "__main__":
    _plot_mvn()


def max_likelihood_linreg(
    fi: np.ndarray,
    targets: np.ndarray,
    lamda: float
) -> np.ndarray:
    '''
    Estimate the maximum likelihood values for the linear model

    Inputs :
    * Fi: [NxM] is the array of basis function vectors
    * t: [Nx1] is the target value for each input in Fi
    * lamda: The regularization constant

    Output: [Mx1], the maximum likelihood estimate of w for the linear model
    '''
    M = fi.shape[1]
    return np.matmul(np.matmul(np.linalg.inv(lamda*np.identity(M) + np.matmul(fi.T,
    fi)), fi.T), targets)

if __name__ == "__main__":
    fi = mvn_basis(X, mu, sigma)  # same as before
    lamda = 0.00051
    wml = max_likelihood_linreg(fi, t, lamda)
    # print(wml)


def linear_model(
    features: np.ndarray,
    mu: np.ndarray,
    sigma: float,
    w: np.ndarray
) -> np.ndarray:
    '''
    Inputs:
    * features: [NxD] is a data matrix with N D-dimensional data vectors.
    * mu: [MxD] matrix of M D dimensional mean vectors defining the
```

```
    multivariate normal distributions.
    * sigma: All normal distributions are isotropic with s*I covariance
    matrices (where I is the MxM identity matrix).
    * w: [Mx1] the weights, e.g. the output from the max_likelihood_linreg
    function.

    Output: [Nx1] The prediction for each data vector in features
    '''
    fi = mvn_basis(features, mu, sigma)
    return np.sum(w*fi, axis=1)

if __name__ == "__main__":
    wml = max_likelihood_linreg(fi, t, lamda)  # as before
    prediction = linear_model(X, mu, sigma, wml)
    print(prediction)
    print(len(prediction), len(t))
    plt.figure("sldk")
    plt.plot(np.array(t), np.array(prediction), '.')
    plt.plot([0, 2],[0,2])
    plt.title("Predictions vs Actual value")
    plt.ylabel("Predictions")
    plt.xlabel("Actual Values")
    plt.savefig("07_linear_models/plot_1_5.png")
    # plt.figure("bars")
    # plt.subplot(2, 1, 1)
    # plt.hist(prediction)
    # plt.subplot(2, 1, 2)
    # plt.hist(t)
    # plt.show()

    def test(M, sigma):
        mu = np.zeros((M, D))
        for i in range(D):
            mmin = np.min(X[:, i])
            mmax = np.max(X[:, i])
            mu[:, i] = np.linspace(mmin, mmax, M)
        # print(N, D, M)
        # print(mu)
        fi = mvn_basis(X, mu, sigma)
        lamda = 0.001
        wml = max_likelihood_linreg(fi, t, lamda)
        prediction = linear_model(X, mu, sigma, wml)
        return np.square(np.subtract(prediction, t)).mean()
    print(f"{test(1000, 7.9)=}")
    Ms = np.array(np.round(np.logspace(1, 3)), np.int32)
    sigmas = np.logspace(-1,2)
    values = np.zeros((100, 100))
    # for i in range(50):
    #     for j in range(50):
    #         print(i, j, Ms[i], sigmas[j])

    #         values[i, j] = test(Ms[i], sigmas[j])
    # with open("07_linear_models/results.npy", 'wb+') as f:
    #     np.save(f, values)

    with open("07_linear_models/results.npy", 'rb') as f:
        results = np.load(f)[:50, :50]

    print(results)

```

```python
    fig, ax = plt.subplots()
    X, Y = np.meshgrid(sigmas, Ms)
    # c = ax.pcolormesh(Ms, sigmas, results, cmap='RdBu',
    #                   vmin=0.04, vmax=0.07, shading="nearest")
    cp = ax.contour(X, Y, results, np.hstack((np.logspace(-1.3, -1, 20)[:4], np.logspace(-1.2, 0.1, 10))))
    ax.set_title(
        'Contour plot of Mean-Square-Errror over \ndifferent combinations of sigma and M')

    ax.set_xlabel("sigma")
    ax.set_ylabel("M")

    # set the limits of the plot to the limits of the data
    # ax.axis([x.min(), x.max(), y.min(), y.max()])
    fig.colorbar(cp, ax=ax)

    print(Ms[results.argmin()//results.shape[1]],
           [results.argmin() % results.shape[1]], "->", np.min(results))
    plt.savefig("07_linear_models/indep1.png")
    plt.show()


### programing sectins done! NOw just need to answer section 1.5 with
# some graphs and text in pdf and do soething for independent section
```