Locally Weighted Regression

J.R. Powers-Luhn

***Abstract*—**

I. Introduction

II. Methodology

A. Body Composition

A dataset of body measurements (body fat percentage, age, weight, height, adiposity index, and ten circumference measurements) was obtained [1]. Ridge regression was used to generate models to predict body fat percentage from the other predictor variables using the two methods described above to select the hyperparameter α.

III. Results

Bandwidth	Order	RMSE (% bodyfat)
0	0.15	11.13
	0.25	11.07
	0.50	8.97
	0.75	7.99
	1.00	7.64
	1.50	7.50
	2.00	7.48
1	0.15	15.64
	0.25	7.53
	0.50	7.48
	0.75	7.95
	1.00	8.07
	1.50	8.08
	2.00	8.14
2	0.15	9.05
	0.25	18.73
	0.50	13.89
	0.75	12.68
	1.00	13.11
	1.50	8.93
	2.00	7.65

A. Model Selection

The model that was selected was that produced by the L-curve method. Its performance was not significantly different from the model produced using the LOO method and the larger α value should result in greater stability. Both models significantly improved on the least squares model, which had an RMSE of 19.35.

The selected model was evaluated on the validation set data, producing an RMSE of 7.78% body fat.

IV. Conclusions

A model was generated to predict body fat percentage from body composition predictors. The error of this model was 7.78% body fat. This was comparable to the error produced by a partial least squares model (4.35%) and showed improvement over principal component regression (4.72%) and a best-guess linear regression (4.79%). Further, the linear model required the calculation of inverse terms while the ridge regression was both straightforward to calculate (unlike PLS, no iterative algorithms were necessary) and did not require non-linear terms. Two methods of selecting the hyperparameter α were examined. Leave one out cross validation was easy to understand but computationally expensive and might be prohibitive for large datasets. Variations on this method (e.g. "Leave K out") are worth exploring if they lead to stable results. The L-curve method of determining α clearly demonstrates the bias/variance trade, making it attractive if potentially subjective.

References

[1] K W Penrose, A G Nelson, and A G Fisher. "Generalized Body Composition Prediction Equation For Men Using Simple Measurement Techniques". In: *Medicine & Science in Sports & Exercise* 17.2 (1985). ISSN: 0195-9131.

V. Appendix

Python code used to perform calculations and generate graphics.

```
import numpy as np
from scipy.stats import norm
from scipy.spatial.distance import cdist
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from utilities import train_test_val_split, load_matlab_data, rmse
from itertools import product
import pandas as pd


x, y = load_matlab_data("data/hwkdataNEW.mat")
xtr, ytr, xts, yts, xv, yv = train_test_val_split(x, y, seed=3)

class LocallyWeightedRegression(BaseEstimator, RegressorMixin):
    def __init__(self, bandwidth=0.25, order=1):
        self.bandwidth = bandwidth
        self.order = order

        # Save the scalers
        self.xscaler_ = None
        self.yscaler_ = None

        # Save training data
        self.xs_ = None
        self.ys_ = None

    def fit(self, x, y):
        self.xscaler_ = StandardScaler().fit(x)
        self.yscaler_ = StandardScaler().fit(y)

        self.xs_ = self.xscaler_.transform(x)
        self.ys_ = self.yscaler_.transform(y)

        return self

    def predict(self, x):
        xs = self.xscaler_.transform(x)
        results = []

        for x in xs:
            # Calculate the distances
            ds = cdist(self.xs_, np.atleast_2d(x))

            # Calculate the weights
            ws = norm.pdf(ds, scale=self.bandwidth)

            # Normalize the weights
            ws /= sum(ws)

            # Convert to 1d array
            ws = ws[:,0]

            # Add in the polynomial terms
            p = PolynomialFeatures(self.order)
```

```
            xt = p.fit_transform(self.xs_)

            # Perform the regression to get the coefficients
            l = LinearRegression()
            l.fit(xt, self.ys_, sample_weight=ws)

            # Calculated the predicted value
            res = l.predict(p.transform(x.reshape((1, -1))))

            # Add that to the results array
            results.append(res)

        res_np = np.array(res).reshape(-1, 1)

        return self.yscaler_.inverse_transform(res_np)

    def score(self, x, y):
        yp = self.predict(x)
        return -rmse(y, yp)

r_dict = {
    'Bandwidth': [],
    'Order': [],
    'RMSE (% bodyfat)': []
}
bandwidths = [0.15, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0]
orders = [0, 1, 2]

params = product(orders, bandwidths)
print("Bandwidth\tOrder\t\tRMSE")
print("-------------------------------------------")
for o, b in params:
    lwr = LocallyWeightedRegression(b, o)
    lwr.fit(xtr, ytr)
    print(f"{b}\t\t{o}\t\t{-lwr.score(xts, yts):0.2f}")
    r_dict['Bandwidth'].append(b)
    r_dict['Order'].append(o)
    r_dict['RMSE (% bodyfat)'].append(f"{-lwr.score(xts, yts):0.2f}")

rdf = pd.DataFrame(r_dict)

with open("table.tab", "w") as f:
    f.write(rdf.to_latex(index=False))

# Validation error
# Best model is first order with bandwidth 0.5
lwr = LocallyWeightedRegression(bandwidth=0.5, order=1)
lwr.fit(xtr, ytr)
print(f"Best model val score: {lwr.score(xv, yv)}")
print(f"{lwr.get_params}")
```