R.D Slater, P.T.C Slater

Modeling Airfoil Data Using Linear Regression

**Introduction**

Airfoil self-noise is caused by the interaction of an airfoil blade and turbulence caused buy the surface of the airfoil as well as the trailing edge. The phenomenon is of interest for various airfoils used in helicopters, wind turbines and airplane wings. NASA Langley Research Center was able to carefully measure both the self noise and various characteristics of airfoil in a published dataset.

**Methods**

Using the NASA published data we attempted to model a linear regression with L2 regularization. The data contains 5 independent variables (Table 1) and 1 target variable. The target variable with the scaled sound pressure level in decibels (dB)

Independent Variables

|  |
| --- |
| Frequency, in Hertz. |
| Angle of attack, in degrees |
| Chord length, in meters. |
| Free-stream velocity, in meters per second |
| Suction side displacement thickness, in meters |

Table 1: A list of the independent variables for the dataset

Data was read into a dataframe and contained 1503 examples with no missing data. The data was then scaled using the StandardScaler from scikit-learn to subtract the mean value and produce a variance of 1 by dividing by the standard deviation.

Once the data was scaled in a 10 fold cross-validation was performed for 20 different strengths of regularization from 10-8 to 10 (evenly chose in log space base 10) and the best performance based on average out of fold mean squared error was chosen to select the best regularization strength. This was repeated 100 times to ensure good confidence intervals. Each split was randomly seeded.

One a final regularization strength was chose a 5 fold cross validation was repeated 100 times to test the out of fold mean squared error to produce an average mean squared error and select 95% Confidence Intervals. The code to conduct this test is attached in Appendix A.

**Results**

We find that regularization only begins to affect the mean squared error at large values of λ. As shown in Figure 1, the mean squared error remains largely unchanged until λ exceeds 0.1.

Chart, bar chart

Description automatically generated

Figure 1: Mean out of fold MSE across 100 runs while varying the strength of regularization. Mean Squared Error loss remains low until 0.1 or larger

For the final λ = 10-8, the 100 runs produce an average Mean Squared Error of 0.5435 with 95% C.I of 0.5422 to 5451.

Chart, histogram

Description automatically generated

Figure 2: The Mean Squared Error for out of fold predictions with a fixed strength of 0.00000001 (the best strength found in hyperparameter search).

**Conclusion**

Our model produced a mean squared error of 0.5435 with 95% C.I of 0.5422 to 5451. The best value of λ was found to be 10-8. Regularization strength produced very little effect below λ = 0.1

Appendix A

import pandas as pd

import numpy as np

from sklearn.linear\_model import Ridge

from sklearn.model\_selection import KFold, cross\_val\_score

from sklearn.preprocessing import StandardScaler

import argparse

import matplotlib.pyplot as plt

def find\_alpha(alphas, model\_, X, y):

    Cs = []

    scores = []

    for i in range(100):

        seed = np.random.randint(0, 100000)

        split = KFold(n\_splits=3, shuffle=True, random\_state=seed)

        best = -1E9

        best\_a = -1

        for a in alphas:

            model\_.alpha = a

            tmp = cross\_val\_score(model\_, X, y, cv=split,

                                  scoring='neg\_mean\_absolute\_error')

            scores.append(-tmp.mean())

            Cs.append(a)

            if tmp.mean() > best:

                best = tmp.mean()

                best\_a = a

    plt.scatter(Cs, scores)

    plt.xscale("log")

    plt.title("Regularization Strength Effect on Mean Squared Error")

    plt.xlabel("Regularization Stength, $\lambda$")

    plt.ylabel("Mean Sqaured Error")

    plt.show()

    return best, best\_a

if \_\_name\_\_ == "\_\_main\_\_":

    parser = argparse.ArgumentParser()

    parser.add\_argument("datafile",

                        help="path to data file")

    args = parser.parse\_args()

    data = pd.read\_csv(args.datafile, sep="\t", header=None)

    scale = StandardScaler()

    scaled\_data = scale.fit\_transform(data)

    model = Ridge()

    candidates = np.logspace(-8, 1, num=20)

    good\_a = find\_alpha(candidates, model, scaled\_data[:, :-1],

                        scaled\_data[:, -1])

    \_, model.alpha = good\_a

    all\_scores = []

    for i in range(100):

        splits = KFold(n\_splits=5, shuffle=True, random\_state=i)

        scores = cross\_val\_score(model,

                                 scaled\_data[:, :-1],

                                 scaled\_data[:, -1],

                                 cv=splits,

                                 scoring='neg\_mean\_absolute\_error')

        all\_scores.append(scores.mean())

    plt.hist(all\_scores)

    plt.xlabel("Mean Sqaured Error")

    plt.title("Range of Scores of $\lambda=10^{-8}$")

    plt.ylabel("Count of Runs")

    plt.show()

    print(np.percentile(all\_scores, 5),

          np.mean(all\_scores),

          np.percentile(all\_scores, 95),

          model.alpha)