[Fish Market Dataset](https://www.kaggle.com/aungpyaeap/fish-market?ref=hackernoon.com" \t "_blank)

Built for multiple linear regression and multivariate analysis, the Fish Market Dataset contains information about common fish species in market sales. The dataset includes the fish species, weight, length, height, and width.

This dataset is a record of 7 common different fish species in fish market sales. With this dataset, a predictive model can be performed using machine friendly data and estimate the weight of fish can be predicted.

Acknowledgements

Thanks to all who make Kernels using this dataset and also people viewed or download this data.

Inspiration

Multiple linear regression is a fundamental practice for this dataset. Multivariate analysis can also be performed.

Data Set of fishes:

# Predict the Weight of Fish

In [ ]:

import copy

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, PolynomialFeatures

from sklearn.linear\_model import LinearRegression, RidgeCV

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

## Data is loaded

In [ ]:

*#The original data can never be added or deleted columns*

original\_data = pd.read\_csv("fish.csv")

*#The data variable is used to make modifications on it*

data = copy.deepcopy(original\_data)

data.head()

linkcode

## Check null data items[¶](https://www.kaggle.com/ivanivanivanivan/predict-the-weight-of-fish-regression#Check-null-data-items)

In [ ]:

np.sum(data.isnull())

There aren't null items

## Data is treated (Strings converted to numerical data)

The only non numerical column is the 'Species', so this one is encoded to an integer

In [ ]:

original\_data["Species"] = pd.DataFrame(original\_data["Species"]).apply(LabelEncoder().fit\_transform)

## Let's see the linear correlation of the different features

In [ ]:

sns.heatmap(data.corr(), annot=True)

In [ ]:

corr = data.corr()["Weight"].drop("Weight")

print(corr)

## Error is studied according to the number of degree of the regression

The aim of this cell is to choose the best degree for the regression. So as to achive this goal:

* It iterates over the diferent degrees.
* For each one, the model is trained several times (50 for example). For each training the training and test error is recorded. For each training, is randomly shuffled between training and test data. The purpose of this strategy is to getting a non random error, by averaging the errors. A conclusion can be drawn from the resulting plots:
* 2 may be the most suitable degree for the regression because:
  + It gets the lowest test error.
  + It gets the closest test error to the training one.
  + That's why a balance between bias and variance is found

In [ ]:

*#Variables for keeping track of errors are initialized*

e\_train = []

e\_test = []

e\_train\_hist = []

e\_test\_hist = []

alpha\_hist = []

alpha = []

*#Max degree of the regression*

max\_degree = 5

*#No. of training times*

training\_times = 50

*#Iterate over the different degrees*

for degree **in** range(1,max\_degree):

poly = PolynomialFeatures(degree)

data = copy.deepcopy(original\_data)

y = pd.DataFrame(data["Weight"])

data = data.drop("Weight", axis = 1)

x = poly.fit\_transform(data)

for i **in** range(training\_times):

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=np.random.randint(100))

model = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1, 2, 4, 6, 8, 16, 32, 40, 50, 80, 100, 150, 200, 250, 300, 350, 400])

model.fit(x\_train, y\_train)

*#Training error is recorded*

e = np.sqrt(mean\_squared\_error(y\_train, model.predict(x\_train)))

e\_train.append(e)

*#Test error is recorded*

e = np.sqrt(mean\_squared\_error(y\_test, model.predict(x\_test)))

e\_test.append(e)

*#The alpha hyperparameter is recorded*

alpha.append(model.alpha\_)

*#The records of the current degree are saved*

e\_train\_hist.append(e\_train)

e\_train = []

e\_test\_hist.append(e\_test)

e\_test = []

alpha\_hist.append(alpha)

alpha = []

*#The mean for each degree is calculated*

e\_train = np.mean(np.array(e\_train\_hist),axis=1)

e\_test = np.mean(np.array(e\_test\_hist),axis=1)

alpha = np.mean(np.array(alpha\_hist),axis=1)

*#The errors and alpha record is plotted*

plt.plot(range(1,max\_degree), e\_train, 'o-', label = "train")

plt.plot(range(1,max\_degree), e\_test, 'o-',label = "test")

plt.legend()

plt.figure()

plt.plot(range(1,max\_degree), alpha, 'o-',label = "alpha")

plt.legend()

## Error is studied according to amount of data

The aim of this cell is to plot the learning curve of the model.  
As a result, it can be easily spotted that training a test error end up close one to each other.  
In addition, the hyperparameter alpha gets bigger and bigger because overfitting is decreasing for every dataset size iteration.

In [ ]:

*#Variables for keeping track of errors are initialized*

e\_train = []

e\_test = []

e\_train\_hist = []

e\_test\_hist = []

alpha\_hist = []

alpha = []

*#Max degree of the regression*

max\_degree = 5

*#No. of training times*

training\_times = 50

*#No. of training examples*

m = original\_data.shape[0]

step = 1

degree = 2

*#For every iteration diferent amounts of data are selected*

for n\_data **in** range(20, m, step):

poly = PolynomialFeatures(degree)

*# The model is trained several times with diferent data so as to get a non-random and more precise error.*

for i **in** range(training\_times):

data = copy.deepcopy(original\_data)

data = data.iloc[np.random.permutation(np.arange(0,m)),:] *#Data is shuffled*

data = data.iloc[1:n\_data,:]

y = pd.DataFrame(data["Weight"])

data = data.drop("Weight", axis = 1)

x = poly.fit\_transform(data)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=np.random.randint(100))

model = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1, 2, 4, 6, 8, 16, 32, 40, 50, 80, 100, 150, 200, 250, 300, 350, 400])

model.fit(x\_train, y\_train)

*#Training error is recorded*

e = np.sqrt(mean\_squared\_error(y\_train, model.predict(x\_train)))

e\_train.append(e)

*#Test error is recorded*

e = np.sqrt(mean\_squared\_error(y\_test, model.predict(x\_test)))

e\_test.append(e)

*#The alpha hyperparameter is recorded*

alpha.append(model.alpha\_)

*#The records of the current degree are saved*

e\_train\_hist.append(e\_train)

e\_train = []

e\_test\_hist.append(e\_test)

e\_test = []

alpha\_hist.append(alpha)

alpha = []

*#The mean for every training examples amount is calculated*

e\_train = np.mean(np.array(e\_train\_hist),axis=1)

e\_test = np.mean(np.array(e\_test\_hist),axis=1)

alpha = np.mean(np.array(alpha\_hist),axis=1)

*#The errors and alpha record are plotted*

plt.plot(range(20, m, step), e\_train, 'o-', label = "train")

plt.plot(range(20, m, step), e\_test, 'o-',label = "test")

plt.legend()

plt.figure()

plt.plot(range(20, m, step), alpha, 'o-',label = "alpha")

plt.legend()

## 

LOGS:

