InstitutoTecnológico y de Estudios Superiores de Occidente

Maestría Ciencia de Datos

Modelado Predictivo

Tarea 1: PLS for regression

Estudiante: Daniel Nuño
Profesor: Dr. Riemann Ruiz Cruz
Fecha entrega: 7 de septiembre 2022

Introduction

The dataset that will be used for this activity has the name **Airfoil Self-Noise Data Set** and it can be found in the repository UC Irvine Machine Learning Repository. The data collected is related to different size NACA 0012 airfoils at various wind tunnel speeds and angles of attack. The span of the airfoil and the observer position were the same in all of the experiment.

Using the data set mentioned, develop the following points.

1. Determine if there is missing data and decide if it is appropriate to use some strategy to fill in the missing data.

2. Create two subsets of data, where the first one will be used for the training process and the second one for the testing process.
3. Train a linear model to estimate the feature “Sound Pressure Level” using as inputs to the model all other variables. Get the values of RMSE and to evaluate the performance of the model in both training and testing.
4. Considering the data set used in point 3; perform elimination of some variables using variance criterion or correlation criterion. With the variables resulting from the elimination process, I trained a new linear model and calculate the metrics RMSE and corresponding to training and testing.
5. Considering the data set used in point 3 again; perform variable reduction by principal component analysis. With the variables resulting from the reduction process, train a new linear model and calculate the metrics RMSE and corresponding to training and testing.
6. Considering the data set used in point 3 again; train a new linear model using PLS technique and calculated the metrics RMSE and corresponding to training and testing.
7. As a result of the previous steps, we have four different linear models to solve the problem proposed. Make a table with the metrics of each model to make a comparison of the models.

Development

In [1]:
```
import pandas as pd
import numpy as np
data = pd.read_csv('airfoil_self_noise.dat', header=None, sep="\s+")
# 1. Frequency, in Hertzs.
# 2. Angle of attack, in degrees.
# 3. Chord length, in meters.
# 4. Free-stream velocity, in meters per second.
# 5. Suction side displacement thickness, in meters.
# 6. Scaled sound pressure level, in decibels. This is the Response variable
names = ["freq", "angle", "clength", "speed", "thickness", "soundp"]
data.columns = names
data.head()
```

Out[1]:

	freq	angle	clength	speed	thickness	soundp
0	800	0.0	0.3048	71.3	0.002663	126.201
1	1000	0.0	0.3048	71.3	0.002663	125.201
2	1250	0.0	0.3048	71.3	0.002663	125.951
3	1600	0.0	0.3048	71.3	0.002663	127.591
4	2000	0.0	0.3048	71.3	0.002663	127.461

Determine if there is missing data and decide if it is appropriate to use some strategy to fill in the missing data.

In [2]:
```
data.describe()
```

Out[2]:

	freq	angle	clength	speed	thickness	soundp
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2886.380572	6.782302	0.136548	50.860745	0.011140	124.835943
std	3152.573137	5.918128	0.093541	15.572784	0.013150	6.898657
min	200.000000	0.000000	0.025400	31.700000	0.000401	103.380000
25%	800.000000	2.000000	0.050800	39.600000	0.002535	120.191000
50%	1600.000000	5.400000	0.101600	39.600000	0.004957	125.721000
75%	4000.000000	9.900000	0.228600	71.300000	0.015576	129.995500
max	20000.000000	22.200000	0.304800	71.300000	0.058411	140.987000

In [3]:
```
data.isnull().sum()
```

Out[3]:
```
freq         0
angle        0
clength      0
speed        0
thickness    0
soundp       0
dtype: int64
```

In [4]:
```
data.isna().sum()
```

Out[4]:
```
freq         0
angle        0
clength      0
speed        0
thickness    0
soundp       0
dtype: int64
```

There are no missing values detected at this point.

Create two subsets of data, where the first one will be used for the training process and the second one for the testing process.

In [5]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data.drop("soundp", axis=1),
                                                    data["soundp"],
                                                    test_size=0.3, random_state=1)
```

Train a linear model to estimate the feature “Sound Pressure Level” using as inputs to the model all other variables. Get the values of RMSE and to evaluate the performance of the model in both training and testing.

In [6]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import (mean_squared_error,r2_score)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_predict_train = linreg.predict(X_train)
y_predict_test = linreg.predict(X_test)

original_mse_train = mean_squared_error(y_train,y_predict_train, squared=False)
original_mse_test = mean_squared_error(y_test,y_predict_test, squared=False)

(original_mse_train, original_mse_test)
```

Out[6]: (4.725472625187291, 4.97823738797826)

Considering the data set used in point 3; perform elimination of some variables using variance criterion or correlation criterion. With the variables resulting from the elimination process, I trained a new linear model and calculate the metrics RMSE and corresponding to training and testing.

In [7]:
```
from sklearn.feature_selection import VarianceThreshold

sel = VarianceThreshold(threshold = 0.5)
sel.fit_transform(X_train)
sel.get_feature_names_out()
```

Out[7]: array(['freq', 'angle', 'speed'], dtype=object)

Accordign to VarianceThreshold, we can keep columns **freq**, **angle** and **speed**. The feature selector that removes all low-variance features. This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

threshold parameter specifies that features with a training-set variance lower than this threshold will be removed. The default is to keep all features with non-zero variance, i.e. remove the features that have the same value in all samples.

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(data[['freq', 'angle', 'speed']],
                                                    data["soundp"],
                                                    test_size=0.3, random_state=1)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_predict_train = linreg.predict(X_train)
y_predict_test = linreg.predict(X_test)

step_mse_train = mean_squared_error(y_train,y_predict_train, squared=False)
step_mse_test = mean_squared_error(y_test,y_predict_test, squared=False)

(step_mse_train, step_mse_test)
```

Out[8]: (5.892135487526167, 5.899347143678915)

Considering the data set used in point 3 again; perform variable reduction by principal component analysis. With the variables resulting from the reduction process, train a new linear model and calculate the metrics RMSE and corresponding to training and testing.

In [9]:
```
from sklearn.decomposition import PCA

X_train, X_test, y_train, y_test = train_test_split(data.drop("soundp", axis=1),
                                                    data["soundp"],
                                                    test_size=0.3, random_state=1)

pca = PCA()
pca.fit(X_train)
data_pca = pca.transform(X_train)
data_pca = pd.DataFrame(data_pca, columns=['x1*','x2*','x3*', 'x4*', 'x5*'])
data_pca["soundp"] = data["soundp"]

pca.explained_variance_ratio_
```

Out[9]:
```
array([9.99973821e-01, 2.30916970e-05, 3.08642017e-06, 6.29162995e-10,
       6.68357489e-12])
```

According to PCA the first principal componenc is suffient because it explains 99.9% of the explained variance.

In [10]:
```
linreg.fit(data_pca[['x1*']], y_train)
y_predict_train = linreg.predict(data_pca[['x1*']])
#pca'd test set
data_pca_test = pca.transform(X_test)
data_pca_test = pd.DataFrame(data_pca_test, columns=['x1*','x2*','x3*', 'x4*', 'x5*'])
linreg.fit(data_pca_test[['x1*']], y_test)
y_predict_test = linreg.predict(data_pca_test[['x1*']])

pca_mse_train = mean_squared_error(y_train,y_predict_train, squared=False)
pca_mse_test = mean_squared_error(y_test,y_predict_test, squared=False)

(pca_mse_train, pca_mse_test)
```

Out[10]:
```
(6.369948882209672, 6.2702984020999315)
```

Considering the data set used in point 3 again; train a new linear model using PLS technique and calculated the metrics RMSE and corresponding to training and testing.

In [11]:
```
from sklearn.cross_decomposition import PLSRegression

X_train, X_test, y_train, y_test = train_test_split(data.drop("soundp", axis=1),
                                                    data["soundp"],
                                                    test_size=0.3, random_state=1)

# Aplicamos PLS
pls = PLSRegression(n_components=1)
pls.fit(data.drop("soundp", axis=1), data["soundp"])
```

```
y_predict_train = pls.predict(X_train)
y_predict_test = pls.predict(X_test)

pls_mse_train = mean_squared_error(y_train,y_predict_train, squared=False)
pls_mse_test = mean_squared_error(y_test,y_predict_test, squared=False)
(pls_mse_train, pls_mse_test)
```

Out[11]: (4.849093920288027, 5.093148922223512)

As a result of the previous steps, we have four different linear models to solve the problem proposed. Make a table with the metrics of each model to make a comparison of the models.

In [12]:
```
results = {'Modelo': ["Original", "Varianza", "PCA", "PLS"],
           'RMSE Train': [original_mse_train, step_mse_train, pca_mse_train, pls_mse_tr
           'RMSE Test': [original_mse_test, step_mse_test, pca_mse_test, pls_mse_test]}

results = pd.DataFrame(data=results)
results
```

Out[12]:

	Modelo	RMSE Train	RMSE Test
0	Original	4.725473	4.978237
1	Varianza	5.892135	5.899347
2	PCA	6.369949	6.270298
3	PLS	4.849094	5.093149

Conclusion

PLS method turns out to be very similar, in termns of error, to the original linear regresión with all data. However with the advantage, or disaventage, to have one liner combined component.