Predictive Modeling

Series 5

Exercise 5.1

Show that

$$P(Y|X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}. \tag{1}$$

and, by setting $p(X) = P(Y=1|X)$,

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X \tag{2}$$

are equivalent.

Exercise 5.2

This exercise has to do with *odds*.

a) For a randomly chosen person in the U.S. the odds of having blue eyes is 0.47. What fraction of people have blue eyes?

b) 25 % of the U.S. citizens have dark brown eyes. What are the odds to encounter a person with dark brown eyes when randomly chosen?

Exercise 5.3

Suppose we collect the following data for the students in the course *predictive modeling*

- X_1 count the hours studied for the final exam
- X_2 is the average mark in the undergraduate studies ("Bachelor")
- Y is 1 if the final mark in the course is 5.5 or higher and 0 else.

Assume further that we have fit a logistic regression model for Y with the coefficients

$$\hat{\beta}_0 = -6, \quad \hat{\beta}_1 = 0.05, \quad \text{and} \quad \hat{\beta}_2 = 1$$

a) Estimate the probability that a student who studies 40h and has an undergraduate mark of 4.5 has a final examination mark of 5.5 or better.

b) How many hours would the student in (a) need to study to a have a 70% chance of getting a mark of 5.5 or better?

Exercise 5.4

We consider again the `Default` data set from the lecture notes where the logistic regression model allowing for the predictors `income`, `balance` and `student` is discussed.

a) Generate a downsampled data set with the same random seed as in the lecture notes (i.e. `np.random.seed(1)`) in order to compare the results with Example 10.3.4 of the lecture notes.

b) Compute a logistic model that only considers `student` as a predictor variable. Estimate the probability that a student/non-student defaults on his/her debt and compare the result with our findings in the example of the multiple logistic regression model `default~balance+income+student` (cf. Example 10.5.1).

c) Generate two boxplots with respect to `balance` for students and non-students and explain the counterintuitive result above.

Exercise 5.5

We will now develop a model to predict whether a given car gets high or low gas mileage based on the `Auto` data set.

a) Get familiar with the Auto data by loading and inspecting the file `auto.csv`.

b) Create a binary variable, `mpg01`, that contains 1 if `mpg` contains a value above its median and 0 else. Create a new data frame that contains `mpg01` and the other `Auto` variables except `mpg` and `name`.

c) Explore the data graphically to get a first idea on which variabels are correlated with `mpg01`. Try scatter- and parallel coordinate plots. The latter can be done with the help of `pd.plotting.parallel_coordinates()`. You may want to scale the different predictors first.

d) Split the data into training and test set.

e) Compute a logistic regression model with the training set. Compute the classification error on the training and the test set and compare it with the cross-validated error.

Exercise 5.6

In this exercise we study an important tool to assess the performance of a binary classifier and to chose the optimal threshold for the class probabilities: the *receiver operating characteristic (ROC)*-curve. We will use the **`Auto`** data from the previous exercise, with **`mpg`** replaced by **`mpg01`**.

a) Use the same training and test set as in the previous exercise and compute the confusion matrix for both sets.

b) Determine the classification accuracy, recall, sensitivity and F1 score on the basis of the test confusion matrix. Consider a 1 as positive result.

c) Write an **`Python`** -function that computes for a given $\alpha \in [0,1]$ and class probabilities the classification result with threshold α.

d) Now loop a threshold value α from 0 to 1 and compute the classification results and store for each α the true positive rate (*recall*) and the false positive rate. Plot the true positive rate against the false positive rate. The resulting curve is called receiver operating characteristic (ROC) curve.
Hint: The *true positive rate* is defined as:

$$\frac{\text{TP}}{\text{TP+FN}}$$

The *false positive rate* is defined as:

$$\frac{\text{FP}}{\text{FP+TN}}$$

e) Interpret the ROC-curve: How can it be used in order to assess the quality of a binary classifier? Give a rule on choosing the optimal threshold parameter.

Result Checker

E 5.2:

a) $p \approx 0.32$

b) $o \approx 0.33$

E 5.3:

a) $p = 0.622$

b) ≈ 47 h

E 5.4:

b) Student: 54.27% , non-Student: 47.69%

E 5.5:

e) Approximate Values:

	training	testing	cross-validation
error	7.96%	11.54%	9.3%

Predictive Modeling

Solutions to Series 5

Solution 5.1 Setting $p(X) = P(Y = 1|X)$ we find from (1) by multiplying both sides with $1 + e^{\beta_0+\beta_1 X}$ that

$$p(X)(1 + e^{\beta_0+\beta_1 X}) = e^{\beta_0+\beta_1 X}.$$

Subtracting $p(X)e^{\beta_0+\beta_1 X}$ then yields

$$p(X) = e^{\beta_0+\beta_1 X} - p(X)e^{\beta_0+\beta_1 X} = e^{\beta_0+\beta_1 X}(1 - p(X)).$$

Dividing by $1 - p(X)$ and taking the log on both sides eventually gives

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X.$$

Solution 5.2 The relation of the odds o and the proportion p has already been deduced in Exercise 1:

$$p = \frac{o}{1+o}, \quad \Leftrightarrow \quad o = \frac{p}{1-p}.$$

a) $p = 0.47/(1 + 0.47) \approx 0.32$.

b) $o = 0.25/(1 - 0.25) \approx 0.33$.

Solution 5.3

a) We estimate the probability by the logistic formula

$$p(X_1, X_2) = \frac{e^{\hat\beta_0+\hat\beta_1 X_1+\hat\beta_2 X_2}}{1 + e^{\hat\beta_0+\hat\beta_1 X_1+\hat\beta_2 X_2}}$$

This gives $p(40, 4.5) = 0.622$.

b) From the log-odds formula we find

$$\log\left(\frac{p(X_1, X_2)}{1 - p(X_1, X_2)}\right) = \hat\beta_0 + \hat\beta_1 X_1 + \hat\beta_2 X_2.$$

For $p(X_1, X_2) = 0.7$ and $X_2 = 4.5$ this gives

$$X_1 = \frac{1}{\hat\beta_1}\left(\log\left(\frac{0.7}{0.3}\right) - \hat\beta_0 - \hat\beta_2 4.5\right) \approx 47\text{h}.$$

Solution 5.4

a) **Python** code:

[1]:
```
import numpy as np
import pandas as pd

# Load data
df = pd.read_csv('./data/Default.csv', sep=';')

# As a first inspection, print the first rows of the data:
print(df.head())
# As well as the dimensions of the set:
print('\nSize of Default =\n', df.shape)
```

```
   Unnamed: 0 default student      balance       income
0           1      No      No   729.526495  44361.62507
1           2      No     Yes   817.180407  12106.13470
2           3      No      No  1073.549164  31767.13895
3           4      No      No   529.250605  35704.49394
4           5      No      No   785.655883  38463.49588

Size of Default =
 (10000, 5)
```

[2]:
```
# Add a numerical column for default and student
df = df.join(pd.get_dummies(df[['default', 'student']],
                            prefix={'default': 'default',
                                    'student': 'student'},
                            drop_first=True))

# Set ramdom seed
np.random.seed(1)
# Index of Yes:
i_yes = df.loc[df['default_Yes'] == 1, :].index

# Random set of No:
i_no = df.loc[df['default_Yes'] == 0, :].index
i_no = np.random.choice(i_no, replace=False, size=333)

i_ds = np.concatenate((i_no, i_yes))

# save downsampled dataframe:
df_ds = df.iloc[i_ds]

# Check dimensions:
print('\nSize of downsampled Default =\n', df_ds.shape)
```

```
Size of downsampled Default =
 (666, 7)
```

b) **Python** code:

[3]:
```
import statsmodels.api as sm

y = df_ds['default_Yes']
x = df_ds['student_Yes']
x_sm = sm.add_constant(x)

model_stud = sm.GLM(y, x_sm, family=sm.families.Binomial())
model_stud = model_stud.fit()

print(model_stud.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            default_Yes   No. Observations:                  666
Model:                            GLM   Df Residuals:                      664
Model Family:                Binomial   Df Model:                            1
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -458.17
Date:                Thu, 25 Mar 2021   Deviance:                       916.34
Time:                        16:39:49   Pearson chi2:                     666.
No. Iterations:                     4
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
const          -0.1444      0.095     -1.517      0.129      -0.331       0.042
student_Yes     0.4347      0.166      2.623      0.009       0.110       0.759
===============================================================================
```

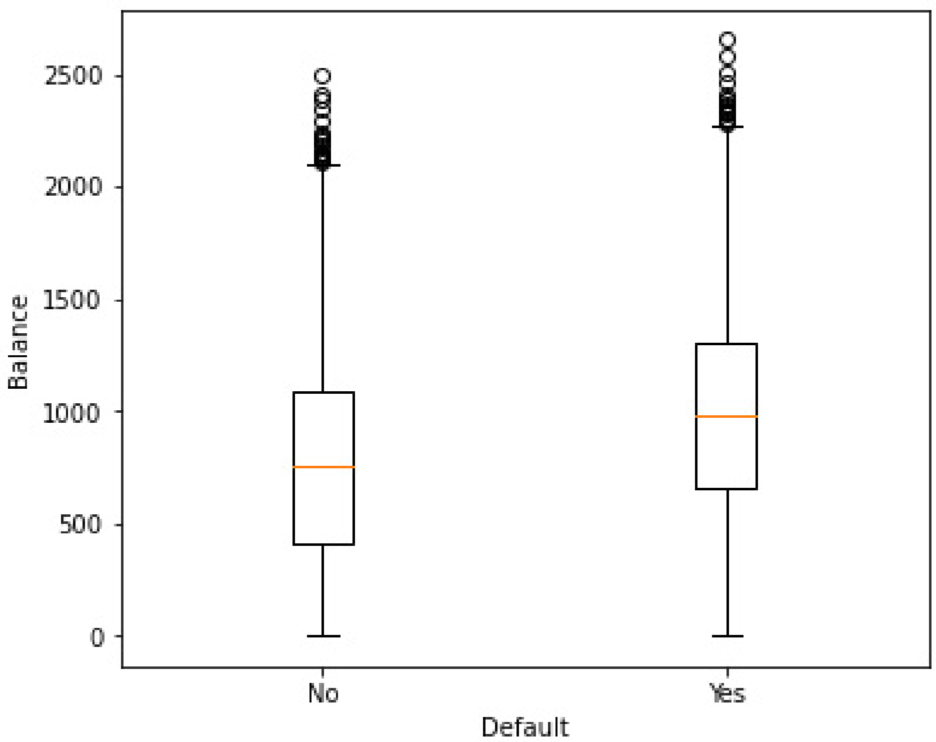
[4]:
```
# Save regression coefficients
beta_0 = model_stud.params[0]
beta_1 = model_stud.params[1]

# Calculate probabilities:
prob_stud = 1 / (1 + np.exp( - (beta_0 + beta_1 * 1)))
prob_nonstud = 1 / (1 + np.exp( - (beta_0 + beta_1 * 0)))
print("probability on default given Student\n",
      np.round(prob_stud, 4),
      "\nprobability on default given not Student\n",
      np.round(prob_nonstud, 4))

# Alternatively, directly predict using .predict():
prob_stud = model_stud.predict([1, 1])
prob_nonstud = model_stud.predict([1, 0])
```

```
probability on default given Student
 0.5721
probability on default given not Student
 0.464
```

It turns out that students are more likely (57.73%) to default on their debt than non-students (46.19%) which can already be seen from the positive coefficient for `student`. This somehow seems to contradict our findings in the multiple logistic regression model from the lecture notes, where the coefficient for `student` was negative.

c) `Python` code:

[6]:
```
import matplotlib.pyplot as plt

# split data based on student status
df_no = df.loc[df['student'] == 'No', :]
df_yes = df.loc[df['student'] == 'Yes', :]

# Create Figure and subplots
fig = plt.figure(figsize=(6, 5))
ax1 = fig.add_subplot(1, 1, 1)
ax1.boxplot([df_no['balance'], df_yes['balance']])
ax1.set_xlabel('Default')
ax1.set_ylabel('Balance')
ax1.set_xticklabels(['No','Yes'])

# plt.tight_layout()
plt.show()
```



The boxplot nicely indicates that the credit card balance is correlated with the student status. Students tend to hold higher levels of debts which is in turn associated with a higher probability of default. This means that an individual student with a given credit card balance will tend to have lower probability to default than a non-student with the same balance. Students *on the whole* hold a higher risk to default because on average they have higher credit card balances, an *individual* student, however, is more trustworthy than a non-student with the same credit card balance.

Solution 5.5

a) Python code:

[1]:
```
import numpy as np
import pandas as pd

# Load data
df = pd.read_csv('./data/auto.csv')

# As a first inspection, print the first rows of the data:
print(df.head())
# As well as the dimensions of the set:
print('\nSize of Auto =\n', df.shape)
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year   0  18.0  ␣
↪     8        307.0         130     3504          12.0    70
1  15.0          8         350.0         165    3693          11.5    70
2  18.0          8         318.0         150    3436          11.0    70
3  16.0          8         304.0         150    3433          12.0    70
4  17.0          8         302.0         140    3449          10.5    70

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino

Size of Auto =
 (392, 9)
```

b) Python code:

[2]:
```
# Create new variable
df['mpg01'] = np.zeros(df.shape[0], dtype=int)
for i in range(df.shape[0]):
    if df.loc[i, 'mpg'] > df['mpg'].median():
        df.loc[i, 'mpg01'] = int(1)
# Drop and add columns
df = df.drop(['mpg', 'name'], axis=1)
```

```
print(df.head())
```

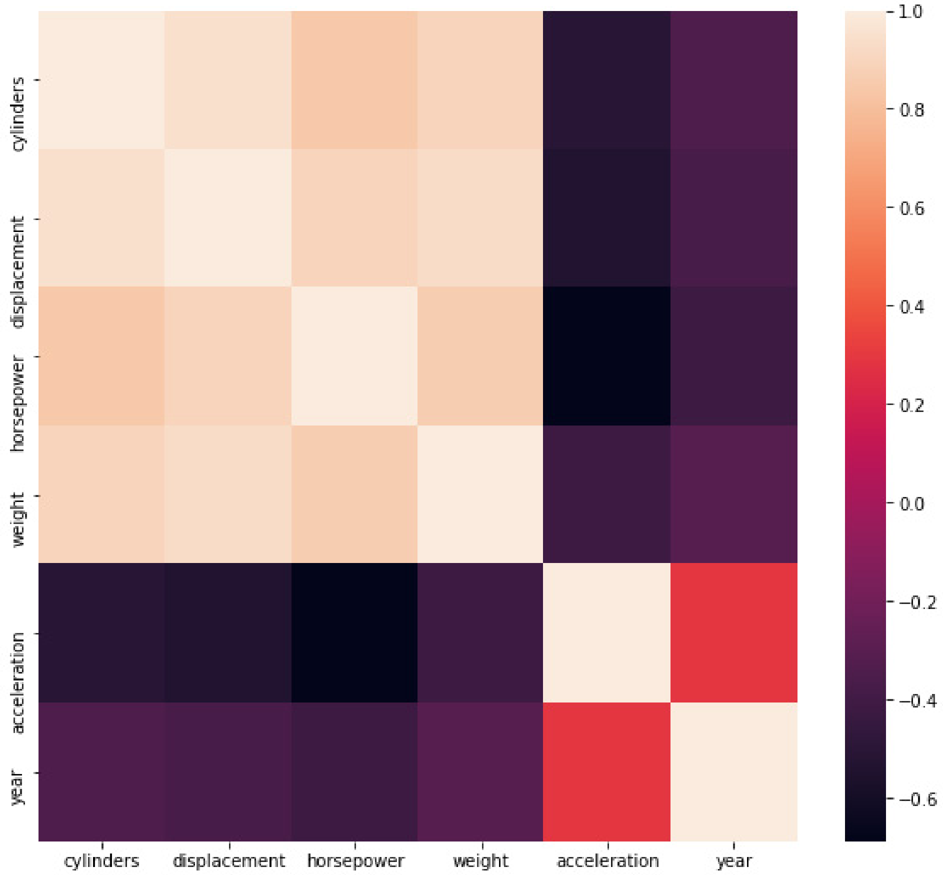
```
   cylinders  displacement  horsepower  weight  acceleration  year  origin   0
↪  8           307.0        130     3504             12.0      70         1
1          8         350.0         165     3693          11.5    70        1
2          8         318.0         150     3436          11.0    70        1
3          8         304.0         150     3433          12.0    70        1
4          8         302.0         140     3449          10.5    70        1

   mpg01
0      0
1      0
2      0
3      0
4      0
```

c) We start by examining the numeric predictors

[3]:
```
""" Heatmap of correlations """
import seaborn as sns
import matplotlib.pyplot as plt

# Find correlations:
corr = df.drop(['origin', 'mpg01'], axis=1).corr()

fig = plt.figure(figsize = (10,8))
ax1 = fig.add_subplot(1, 1, 1)

sns.heatmap(corr)

plt.show()
```

[4]:
```
""" Pairplots: """
fig = sns.pairplot(df.drop(['origin', 'mpg01'], axis=1))
plt.show()
```

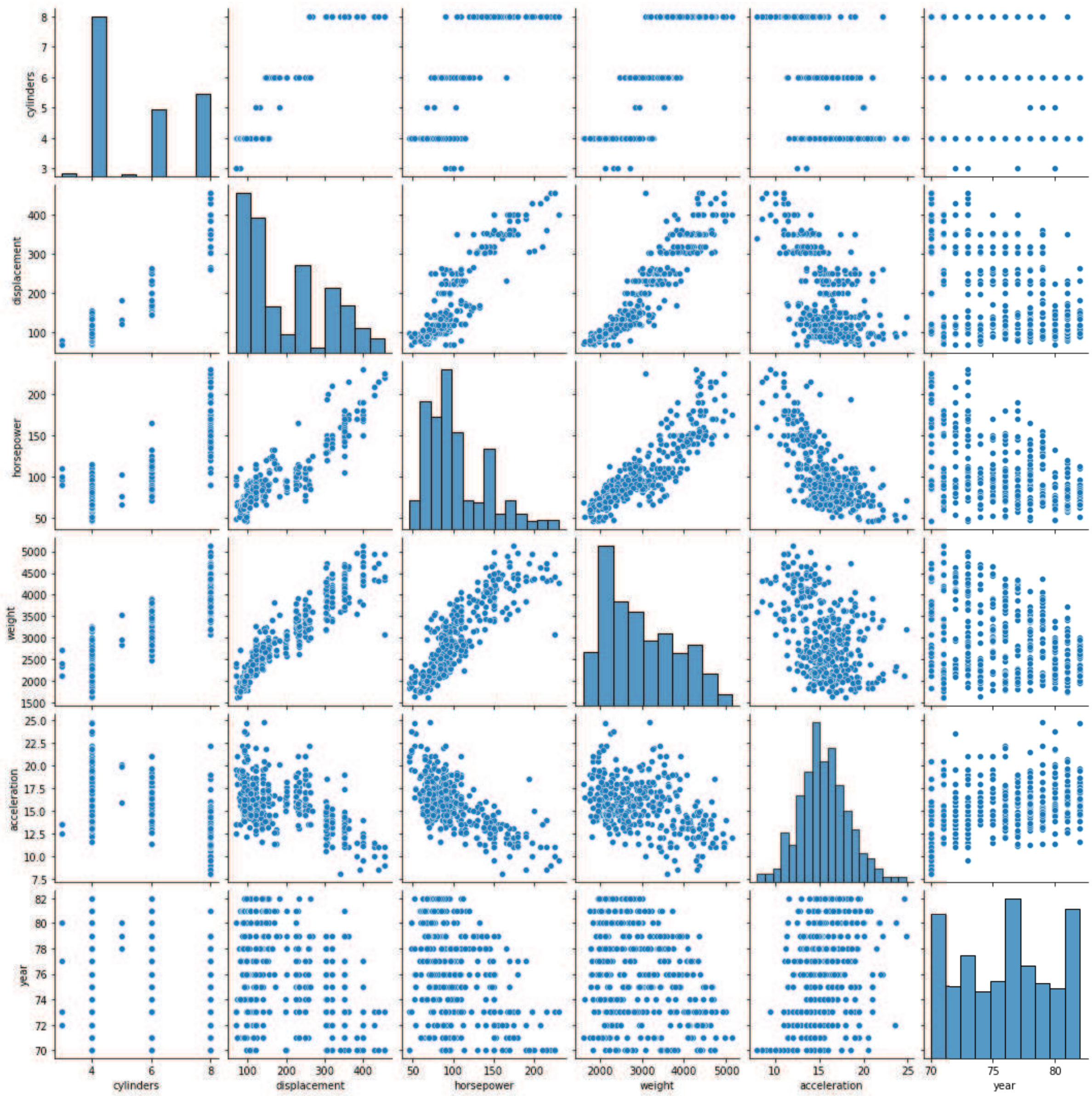
[5]:
```
""" Parallel coordinates using Pandas """
# Option 1, manualy scaling:
df_nor = df.drop(['origin'], axis=1).copy()
for col in df_nor.columns.values:
    df_nor[col] = df_nor[col] - df_nor[col].min()
    df_nor[col] = df_nor[col] / (df_nor[col].max() - df_nor[col].min())

# Option 2, scaling using sklearn:
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_nor = df.drop(['origin'], axis=1).copy()

df_nor[df_nor.columns] = scaler.fit_transform(df_nor[df_nor.columns] )
```




```
# Plot parallel coordinates:
fig = plt.figure(figsize = (14,6))
ax = fig.add_subplot(1, 1, 1)
pd.plotting.parallel_coordinates(df_nor, 'mpg01',
                                 ax=ax, color=('k', 'r'))
plt.show()
```

According to the scatter and parallel coordinate plots, we observe that the two classes are separable (according to the concentration of data points with the same color). These plots indicate that there might be good predictors for **mpg01** in the data set. The correlation between the fuel consumption and the origin of the car can be studied by a contingency table

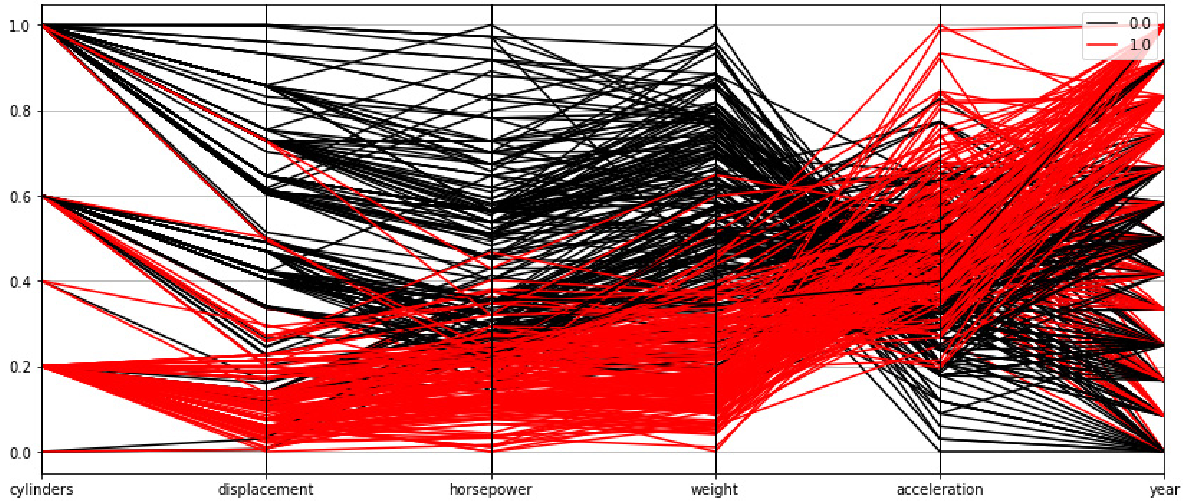
[1]:
```
# Create matrix
confusion = pd.DataFrame({'mpg01': df['mpg01'],
                          'origin': df['origin']})
confusion = pd.crosstab(confusion.origin, confusion.mpg01,
                        margins=True, margins_name="Sum")
```




```
print(confusion)
```

```
mpg01      0    1  Sum
origin
1        173   72  245
2         14   54   68
3          9   70   79
Sum      196  196  392
```

The table tells us, that Japanese and European cars in the data set have a high mileage compared to the american cars. In particular, **mpg01** and **origin** are highly correlated.



d) We randomly split the data in a training and a test set. We choose a 4 : 1 ratio

[7]:
```
# Redefine origin as a categorical variable
df = pd.get_dummies(data=df, drop_first=False,
                    columns=['origin'])
df = df.rename(columns={'origin_1': 'American',
                       'origin_2': 'European',
                       'origin_3': 'Japanese'})
```

[8]:
```
# Set ramdom seed
np.random.seed(2)

i = df.index
# Index of test
i_test = np.random.choice(i, replace=False,
                          size=int(df.shape[0] / 5))

# Save DataFrames
df_test = df.iloc[i_test]
df_train = df.drop(i_test)

# Check dimensions:
print('\nSize of Train =\n', df_train.shape,
      '\nSize of Test =\n', df_test.shape)
```

```
Size of Train =
 (314, 10)
Size of Test =
 (78, 10)
```

e) We first compute the model and examine the coefficients

[1]:
```
import statsmodels.api as sm

y = df_train['mpg01']
x = df_train.drop(['mpg01'], axis=1)
x_sm = sm.add_constant(x)

model = sm.GLM(y, x_sm, family=sm.families.Binomial())
model = model.fit()

print(model.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                  mpg01   No. Observations:                  314
Model:                            GLM   Df Residuals:                      305
Model Family:                Binomial   Df Model:                            8
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -54.944
Date:                Fri, 26 Mar 2021   Deviance:                       109.89
Time:                        11:56:34   Pearson chi2:                     178.
No. Iterations:                     8
Covariance Type:            nonrobust
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const          -14.8950      5.364     -2.777      0.005     -25.408      -4.382
cylinders       -0.5266      0.512     -1.028      0.304      -1.531       0.478
displacement     0.0320      0.016      2.007      0.045       0.001       0.063
horsepower      -0.0538      0.029     -1.828      0.068      -0.112       0.004
weight          -0.0069      0.002     -4.015      0.000      -0.010      -0.004
acceleration    -0.0144      0.176     -0.082      0.935      -0.359       0.330
year             0.5627      0.113      4.997      0.000       0.342       0.783
American        -6.7099      1.978     -3.392      0.001     -10.587      -2.833
European        -3.8513      1.753     -2.197      0.028      -7.286      -0.416
Japanese        -4.3338      1.839     -2.357      0.018      -7.938      -0.730
================================================================================
```

We see that **weight** and **year** are significant factors in the model. We next compute the classification errors

[10]:
```
# Predict for train and test
def class_err(x, y, model):
    """ Find classification error for given
    x, y and fitted model """
    y_pred = model.predict(x)
    # Round to 0 or 1
    y_pred = y_pred.round()
    # Classification error
    e = abs(y - y_pred).mean()
    return e

y_test = df_test['mpg01']
x_test = df_test.drop(['mpg01'], axis=1)
x_sm_test = sm.add_constant(x_test)
```

```
e_train = class_err(x_sm, y, model)
e_test = class_err(x_sm_test, y_test, model)

print('Train error:\n', np.round(e_train, 4),
      '\nTest error:\n', np.round(e_test, 4))
```

```
Train error:
 0.0764
Test error:
 0.1154
```

Solution 5.6

a) `Python` code:

[1]:
```
import numpy as np
import pandas as pd

# Load data
df = pd.read_csv('./data/auto.csv')
# Create new variable
df['mpg01'] = np.zeros(df.shape[0], dtype=int)
for i in range(df.shape[0]):
    if df.loc[i, 'mpg'] > df['mpg'].median():
        df.loc[i, 'mpg01'] = int(1)
# Drop and add columns
df = df.drop(['mpg', 'name'], axis=1)
# Redefine origin as a categorical variable
df = pd.get_dummies(data=df, drop_first=False,
                    columns=['origin'])
df = df.rename(columns={'origin_1': 'American',
                   'origin_2': 'European',
                   'origin_3': 'Japanese'})
# Set ramdom seed
np.random.seed(2)

# Index of test
i_test = np.random.choice(df.index, replace=False,
                          size=int(df.shape[0] / 5))

# Save DataFrames
df_test = df.iloc[i_test]
df_train = df.drop(i_test)

# Check dimensions:
print('\nSize of Train =\n', df_train.shape,
      '\nSize of Test =\n', df_test.shape)
```

```
Size of Train =
 (314, 10)
```

```
Size of Test =
 (78, 10)
```

[2]:

```python
import statsmodels.api as sm

y_train = df_train['mpg01']
y_test = df_test['mpg01']

x_train = df_train.drop(['mpg01'], axis=1)
x_test = df_test.drop(['mpg01'], axis=1)

x_sm_train = sm.add_constant(x_train)
x_sm_test = sm.add_constant(x_test)

# Create and fit logistic regression model
model = sm.GLM(y_train, x_sm_train,
               family=sm.families.Binomial())
model = model.fit()

# Predict on train and testset
y_pred_train = model.predict(x_sm_train).round()
y_pred_test = model.predict(x_sm_test).round()

# Create confusion matrix
confusion_train = pd.DataFrame({'predicted': y_pred_train,
                                'true': y_train})
confusion_test = pd.DataFrame({'predicted': y_pred_test,
                                'true': y_test})
confusion_train = pd.crosstab(confusion_train.predicted,
                              confusion_train.true,
                              margins=True, margins_name="Sum")
confusion_test = pd.crosstab(confusion_test.predicted,
                              confusion_test.true,
                              margins=True, margins_name="Sum")

print(confusion_test, '\n\n',
      confusion_train)
```

```
true        0   1  Sum
predicted
0.0        35   3   38
1.0         6  34   40
Sum        41  37   78

 true         0    1  Sum
predicted
0.0        141   10  151
1.0         14  149  163
Sum        155  159  314
```

b) **Python** code:

[3]:
```
# Accuracy : (tp + tn) / (tp + fp + fn + tn )
tp = confusion_test[1][1]
tn = confusion_test[0][0]
fp = confusion_test[1][0]
fn = confusion_test[0][1]
Accuracy = (tp + tn) / (tp + fp + fn + tn )
print(np.round(Accuracy, 4))
```

```
0.8846
```

[4]:
```
# Precision: tp / (tp + fp)
Precision = tp / (tp + fp)
print(np.round(Precision, 4))
```

```
0.9189
```

[5]:
```
# Recall: tp / (tp + fn)
Recall = tp / (tp + fn)
print(np.round(Recall, 4))
```

```
0.85
```

[6]:
```
# F1-Score: 2 * precision * recall / (precision +  recall)
F1_Score = 2 * Precision * Recall / (Precision +  Recall)
print(np.round(F1_Score, 4))
```

```
0.8831
```

c) **Python** code:

[7]:
```
def class_a(alpha, probability):
    classification = np.zeros(len(probability), dtype=int)
    for i in range(len(probability)):
        if probability.iloc[i] > alpha:
            classification[i] = 1

    return classification
```

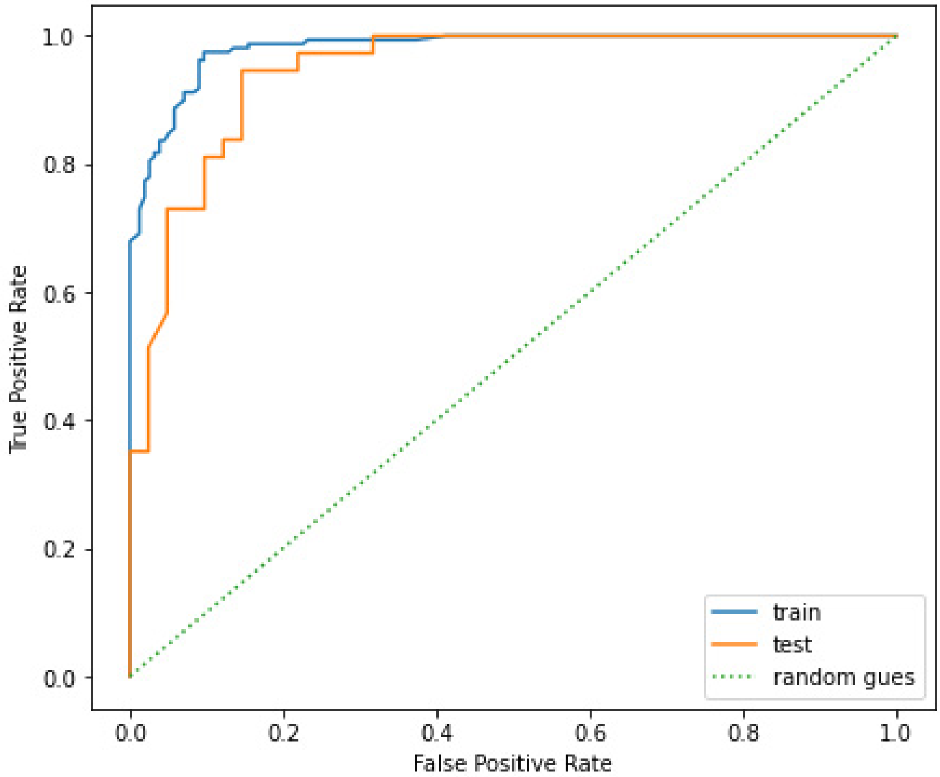
d) **Python** code:

[1]:
```
import matplotlib.pyplot as plt

n = 100

alpha = np.linspace(0, 1, n)

# Create defintion returning both recall and fpr:
```

```
def ROC_data(x, y, model, alpha):
    """ Return Recall and False Posite Rate
    for a given x, y, model and threshold alpha """
    y_pred = class_a(alpha, model.predict(x))

    tp = (y_pred[y_pred == y] == 1).sum()
    tn = (y_pred[y_pred == y] == 0).sum()
    fp = (y_pred[y_pred != y] == 1).sum()
    fn = (y_pred[y_pred != y] == 0).sum()
    # Recall: tp / (tp + fn)
    Recall = tp / (tp + fn)
    fpr = fp / (fp + tn)

    return fpr, Recall

fpr_train, Recall_train = np.zeros(n), np.zeros(n)
fpr_test, Recall_test = np.zeros(n), np.zeros(n)

for i in range(n):
    fpr_train[i], Recall_train[i] = (ROC_data(
        x_sm_train, y_train, model, alpha[i]))
    fpr_test[i], Recall_test[i] = (ROC_data(
        x_sm_test, y_test, model, alpha[i]))

""" Plot ROC curve """
fig = plt.figure(figsize = (7,6))
ax = fig.add_subplot(1, 1, 1)

plt.plot(fpr_train, Recall_train, label='train')
plt.plot(fpr_test, Recall_test, label='test')
plt.plot([0, 1], [0, 1], ':', label='random gues')
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
plt.legend()
plt.show()
```

e) Each binary classifier corresponds to a point in the ROC-space (the unit square). The left upper corner of the ROC-space contains the best classifiers for they have a true positive rate of 100% and a false positive rate of 0. The diagonal line corresponds to random classifiers. So for the classification method *on the whole* it is best to have a ROC-curve that runs as close as possible to the left and upper edge of the square. The area under curve (AUC) is often used to measure the quality of the method. The closest point on the curve to $(0, 1)$ then corresponds to the best threshold and in turn to the best classifier.

[9]:
```
# AUC by right riemann sum
AUC_train, AUC_test = 0, 0
for i in range(n-1):
```




```
    AUC_train += Recall_train[i] * (fpr_train[i] - fpr_train[i + 1])
    AUC_test += Recall_test[i] * (fpr_test[i] - fpr_test[i + 1])

print("AUC train:\n", np.round(AUC_train, 4),
      "\nAUC test:\n", np.round(AUC_test, 4))
```

```
AUC train:
 0.9813
AUC test:
 0.9453
```

[10]:

```
# Best classifier:
dist_train, dist_test = [], []
for i in range(n):
    dist_train.append(fpr_train[i]**2 + (1 - Recall_train[i])**2)
    dist_test.append(fpr_test[i]**2 + (1 - Recall_test[i])**2)

alpha_train = alpha[np.argmin(dist_train)]
alpha_test = alpha[np.argmin(dist_test)]

print("Best alpha according to train data:\n",
      np.round(alpha_train, 3),
      "\nBest alpha according to test data:\n",
      np.round(alpha_test, 3))
```

```
Best alpha according to train data:
 0.475
Best alpha according to test data:
 0.313
```