Machine Learning TA2
Bayesian Classifier Implementation

You-Xun Chang
kevinchang0526@m111.nthu.edu.tw

April 9, 2024

[1]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
# import seaborn as sns
```

1 Load Data

[2]:
```
Auto = pd.read_csv("Auto.csv")
Auto
```

[2]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0       130.0    3504          12.0    70
1    15.0          8         350.0       165.0    3693          11.5    70
2    18.0          8         318.0       150.0    3436          11.0    70
3    16.0          8         304.0       150.0    3433          12.0    70
4    17.0          8         302.0       140.0    3449          10.5    70
..    …          …             …           …      …             …    …
392  27.0          4         140.0        86.0    2790          15.6    82
393  44.0          4          97.0        52.0    2130          24.6    82
394  32.0          4         135.0        84.0    2295          11.6    82
395  28.0          4         120.0        79.0    2625          18.6    82
396  31.0          4         119.0        82.0    2720          19.4    82

     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
..      …                          …
392       1            ford mustang gl
393       2                  vw pickup
394       1              dodge rampage
395       1                ford ranger
396       1                 chevy s-10

[397 rows x 9 columns]
```

[3]:
```
Auto.head(n=10)
```

[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino
5	15.0	8	429.0	198.0	4341	10.0	70	1	ford galaxie 500
6	14.0	8	454.0	220.0	4354	9.0	70	1	chevrolet impala
7	14.0	8	440.0	215.0	4312	8.5	70	1	plymouth fury iii
8	14.0	8	455.0	225.0	4425	10.0	70	1	pontiac catalina
9	15.0	8	390.0	190.0	3850	8.5	70	1	amc ambassador dpl

Dataset description

- mpg: miles per gallon
- cylinders: Number of cylinders between 4 and 8
- displacement: Engine displacement (cu. inches)
- horsepower: Engine horsepower
- weight: Vehicle weight (lbs.)
- acceleration: Time to accelerate from 0 to 60 mph (sec.)
- year: Model year (modulo 100)
- origin: Origin of car (1. American, 2. European, 3. Japanese)
- name: Vehicle name

1.1 Data Exploration / Data Proprocessing

[4]:
```
### Check NaN
Auto.isna().sum(axis=0)
```

[4]:
```
mpg             0
cylinders       0
displacement    0
horsepower      5
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

[5]:
```
### Select Column
Auto.loc[:,'mpg']
```

[5]:
```
0      18.0
1      15.0
2      18.0
3      16.0
4      17.0
       …
392    27.0
393    44.0
394    32.0
395    28.0
396    31.0
Name: mpg, Length: 397, dtype: float64
```

[6]:
```
### Select Data
Auto.iloc[30:35,:]
```

[6]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
30	28.0	4	140.0	90.0	2264	15.5	71	1	chevrolet vega 2300
31	25.0	4	113.0	95.0	2228	14.0	71	3	toyota corona
32	25.0	4	98.0	NaN	2046	19.0	71	1	ford pinto
33	19.0	6	232.0	100.0	2634	13.0	71	1	amc gremlin
34	16.0	6	225.0	105.0	3439	15.5	71	1	plymouth satellite custom

[7]:
```
### Delete Row
Auto = Auto.drop(32, axis=0)
Auto.iloc[30:35,:]
```

[7]:
```
     mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
30  28.0          4         140.0        90.0    2264          15.5    71
31  25.0          4         113.0        95.0    2228          14.0    71
33  19.0          6         232.0       100.0    2634          13.0    71
34  16.0          6         225.0       105.0    3439          15.5    71
35  17.0          6         250.0       100.0    3329          15.5    71

    origin                       name
30       1        chevrolet vega 2300
31       3              toyota corona
33       1                amc gremlin
34       1  plymouth satellite custom
35       1  chevrolet chevelle malibu
```

[8]:
```
### Fetch new column variable
Auto_name = Auto.loc[:,'name']
Auto_name
```

[8]:
```
0      chevrolet chevelle malibu
1              buick skylark 320
2             plymouth satellite
3                  amc rebel sst
4                    ford torino
                 …
392              ford mustang gl
393                    vw pickup
394                dodge rampage
395                  ford ranger
396                   chevy s-10
Name: name, Length: 396, dtype: object
```

[9]:
```
### Drop Column
Auto = Auto.drop('name', axis=1)
Auto
```

[9]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0       130.0    3504          12.0    70
1    15.0          8         350.0       165.0    3693          11.5    70
2    18.0          8         318.0       150.0    3436          11.0    70
3    16.0          8         304.0       150.0    3433          12.0    70
4    17.0          8         302.0       140.0    3449          10.5    70
..    …          …           …          …        …             …    …
392  27.0          4         140.0        86.0    2790          15.6    82
```

```
393  44.0          4           97.0        52.0    2130          24.6    82
394  32.0          4          135.0        84.0    2295          11.6    82
395  28.0          4          120.0        79.0    2625          18.6    82
396  31.0          4          119.0        82.0    2720          19.4    82

     origin
0         1
1         1
2         1
3         1
4         1
..      …
392       1
393       2
394       1
395       1
396       1

[396 rows x 8 columns]
```

[10]:
```
### Concat Data
Auto = pd.concat([Auto, Auto_name], axis=1)
Auto
```

[10]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0       130.0    3504          12.0    70
1    15.0          8         350.0       165.0    3693          11.5    70
2    18.0          8         318.0       150.0    3436          11.0    70
3    16.0          8         304.0       150.0    3433          12.0    70
4    17.0          8         302.0       140.0    3449          10.5    70
..    …          …             …           …      …             …    …
392  27.0          4         140.0        86.0    2790          15.6    82
393  44.0          4          97.0        52.0    2130          24.6    82
394  32.0          4         135.0        84.0    2295          11.6    82
395  28.0          4         120.0        79.0    2625          18.6    82
396  31.0          4         119.0        82.0    2720          19.4    82

     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
..      …                          …
392       1            ford mustang gl
393       2                  vw pickup
394       1              dodge rampage
```

```
395       1                ford ranger
396       1                 chevy s-10

[396 rows x 9 columns]
```

[11]:
```
### Drop All NaN
Auto = Auto.dropna()
Auto = Auto.drop('name', axis=1)
Auto
```

[11]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0       130.0    3504          12.0    70
1    15.0          8         350.0       165.0    3693          11.5    70
2    18.0          8         318.0       150.0    3436          11.0    70
3    16.0          8         304.0       150.0    3433          12.0    70
4    17.0          8         302.0       140.0    3449          10.5    70
..    …          …           …          …        …             …     …
392  27.0          4         140.0        86.0    2790          15.6    82
393  44.0          4          97.0        52.0    2130          24.6    82
394  32.0          4         135.0        84.0    2295          11.6    82
395  28.0          4         120.0        79.0    2625          18.6    82
396  31.0          4         119.0        82.0    2720          19.4    82

     origin
0         1
1         1
2         1
3         1
4         1
..      …
392       1
393       2
394       1
395       1
396       1

[392 rows x 8 columns]
```

[12]:
```
### Check Data
print('Counts of NaN value: \n', Auto.isna().sum(axis=0), '\n')
print('Column data type: \n', Auto.dtypes)
```

```
Counts of NaN value:
 mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
```

```
acceleration    0
year            0
origin          0
dtype: int64

Column data type:
 mpg             float64
cylinders          int64
displacement     float64
horsepower       float64
weight             int64
acceleration     float64
year               int64
origin             int64
dtype: object
```

[13]:
```
### Add new binary label
Auto['mpg_label'] = ''
display(Auto.head(n=10))
mpg_mean = np.mean(Auto.loc[:,'mpg'])
Auto.loc[Auto.loc[:,'mpg']>=mpg_mean, 'mpg_label'] = 'high'
Auto.loc[~(Auto.loc[:,'mpg']>=mpg_mean), 'mpg_label'] = 'low'
display(Auto)
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0       130.0    3504          12.0    70
1  15.0          8         350.0       165.0    3693          11.5    70
2  18.0          8         318.0       150.0    3436          11.0    70
3  16.0          8         304.0       150.0    3433          12.0    70
4  17.0          8         302.0       140.0    3449          10.5    70
5  15.0          8         429.0       198.0    4341          10.0    70
6  14.0          8         454.0       220.0    4354           9.0    70
7  14.0          8         440.0       215.0    4312           8.5    70
8  14.0          8         455.0       225.0    4425          10.0    70
9  15.0          8         390.0       190.0    3850           8.5    70

   origin mpg_label
0       1
1       1
2       1
3       1
4       1
5       1
6       1
7       1
8       1
9       1

      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
```

```
0    18.0          8         307.0       130.0    3504          12.0    70
1    15.0          8         350.0       165.0    3693          11.5    70
2    18.0          8         318.0       150.0    3436          11.0    70
3    16.0          8         304.0       150.0    3433          12.0    70
4    17.0          8         302.0       140.0    3449          10.5    70
..    …          …           …          …        …             …    …
392  27.0          4         140.0        86.0    2790          15.6    82
393  44.0          4          97.0        52.0    2130          24.6    82
394  32.0          4         135.0        84.0    2295          11.6    82
395  28.0          4         120.0        79.0    2625          18.6    82
396  31.0          4         119.0        82.0    2720          19.4    82

     origin mpg_label
0         1       low
1         1       low
2         1       low
3         1       low
4         1       low
..      …       …
392       1      high
393       2      high
394       1      high
395       1      high
396       1      high

[392 rows x 9 columns]
```

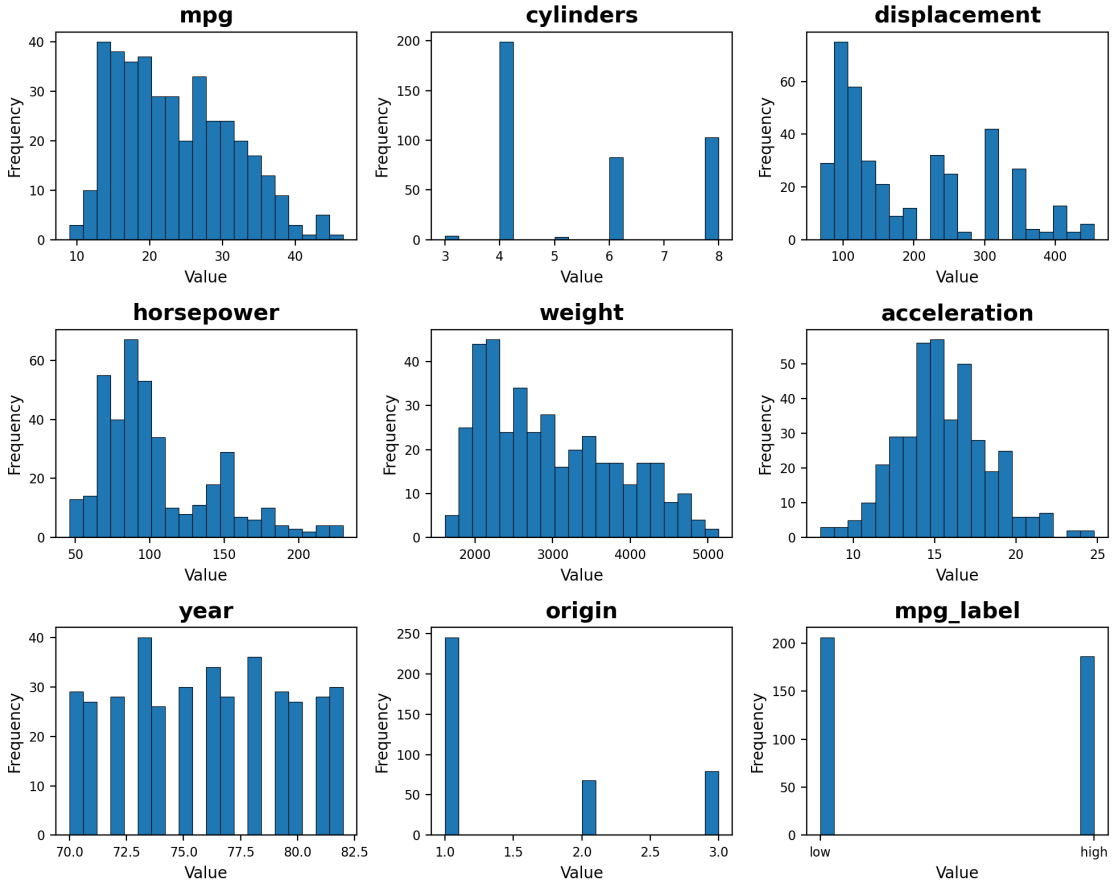
[14]:
```
# Label encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(Auto['mpg_label'])
Auto['mpg_label_01'] = le.transform(Auto['mpg_label'])
display(Auto)
```

```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0       130.0    3504          12.0    70
1    15.0          8         350.0       165.0    3693          11.5    70
2    18.0          8         318.0       150.0    3436          11.0    70
3    16.0          8         304.0       150.0    3433          12.0    70
4    17.0          8         302.0       140.0    3449          10.5    70
..    …          …           …          …        …             …    …
392  27.0          4         140.0        86.0    2790          15.6    82
393  44.0          4          97.0        52.0    2130          24.6    82
394  32.0          4         135.0        84.0    2295          11.6    82
395  28.0          4         120.0        79.0    2625          18.6    82
396  31.0          4         119.0        82.0    2720          19.4    82

     origin mpg_label  mpg_label_01
```

```
0          1        low             1
1          1        low             1
2          1        low             1
3          1        low             1
4          1        low             1
..       …        …             …
392        1       high             0
393        2       high             0
394        1       high             0
395        1       high             0
396        1       high             0

[392 rows x 10 columns]
```

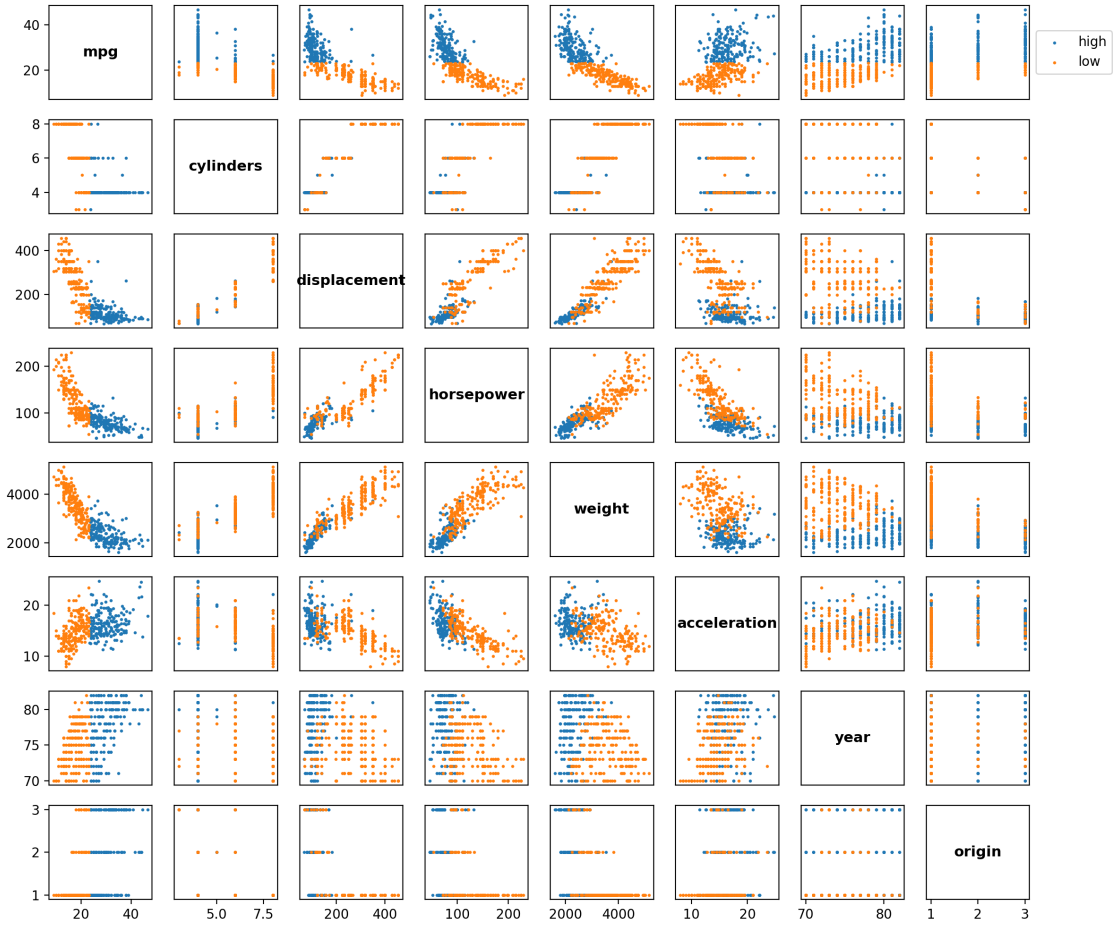
1.2 Visualization

[15]:
```
### Visualization(1) - Histogram
plt.figure(figsize = (10,8), dpi=200)
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.hist(Auto.iloc[:,i], bins=20, rwidth=1, edgecolor='black', linewidth=0.
 ↪4)
    plt.xticks(fontsize=8)
    plt.yticks(fontsize=8)
    plt.xlabel('Value', fontsize=10)
    plt.ylabel('Frequency', fontsize=10)
    plt.title(Auto.columns[i], fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



[16]:
```
### Visualization(2) - Scatter Plot
plt.figure(figsize = (12,10), dpi=200)
for i in range(8):
```

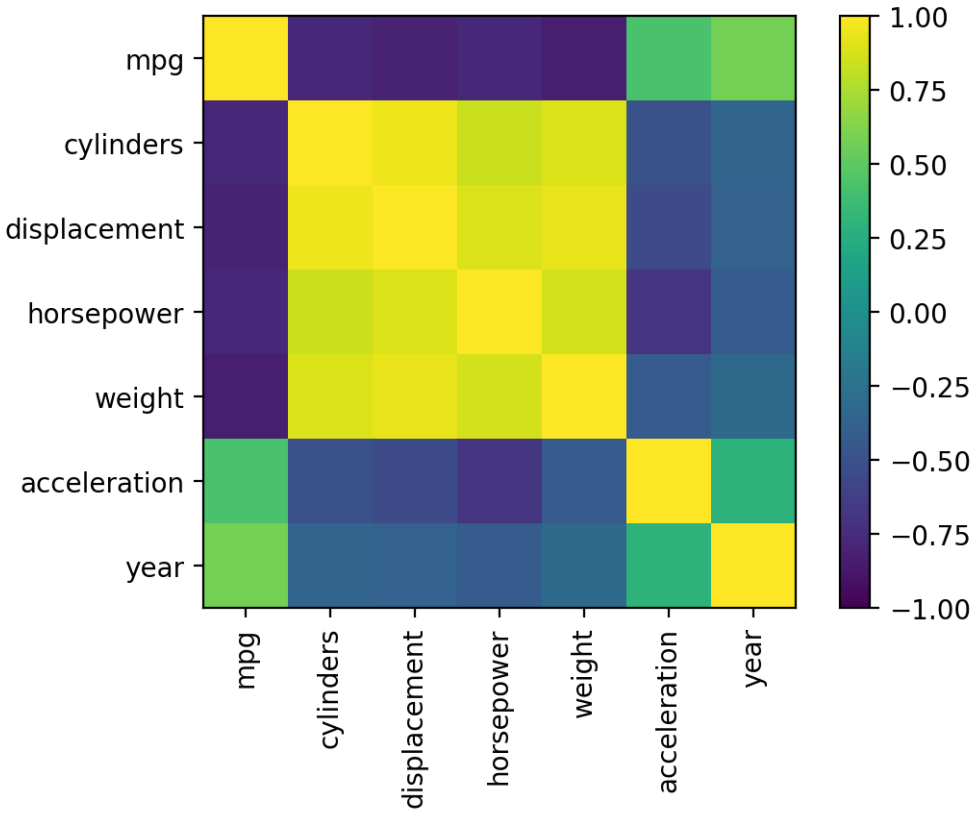
```
    for j in range(8):
        plt.subplot(8,8,i*8+j+1)
        if i!=j:
            plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'high').values,j],Auto.
 ↪iloc[(Auto.iloc[:,8] == 'high').values,i], s=1.8)
            plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'low').values,j],Auto.
 ↪iloc[(Auto.iloc[:,8] == 'low').values,i], s=1.8)
            if i==0:
                if j==7:
                    plt.legend(['high', 'low'], loc='center␣
 ↪left',bbox_to_anchor=(1, 0.5))
        if i==j:
            x_loc = np.mean([np.min(Auto.iloc[:,j]), np.max(Auto.iloc[:,j])])
            y_loc = np.mean([np.min(Auto.iloc[:,i]), np.max(Auto.iloc[:,i])])
            plt.scatter(Auto.iloc[:,j],Auto.iloc[:,i], s=0)
            plt.text(y_loc, x_loc, Auto.columns[i],␣
 ↪horizontalalignment='center', verticalalignment='center',
                     fontsize=11, fontweight='bold')
        if j!=0:
            plt.yticks([])
        if i!=7:
            plt.xticks([])
plt.tight_layout()
#sns.pairplot(Auto.iloc[:,0:7], diag_kind=None)
plt.show()
```



[17]:
```
### Correlation
corrM = Auto.iloc[:,0:7].corr()
display(round(corrM,3))

plt.figure(figsize = (6,4), dpi=200)
plt.imshow(corrM)
plt.colorbar()
plt.clim([-1,1])
plt.xticks(np.arange(7), corrM.columns[:], rotation='vertical')
plt.yticks(np.arange(7), corrM.columns[:])
#sns.heatmap(corrM)
plt.show()
```

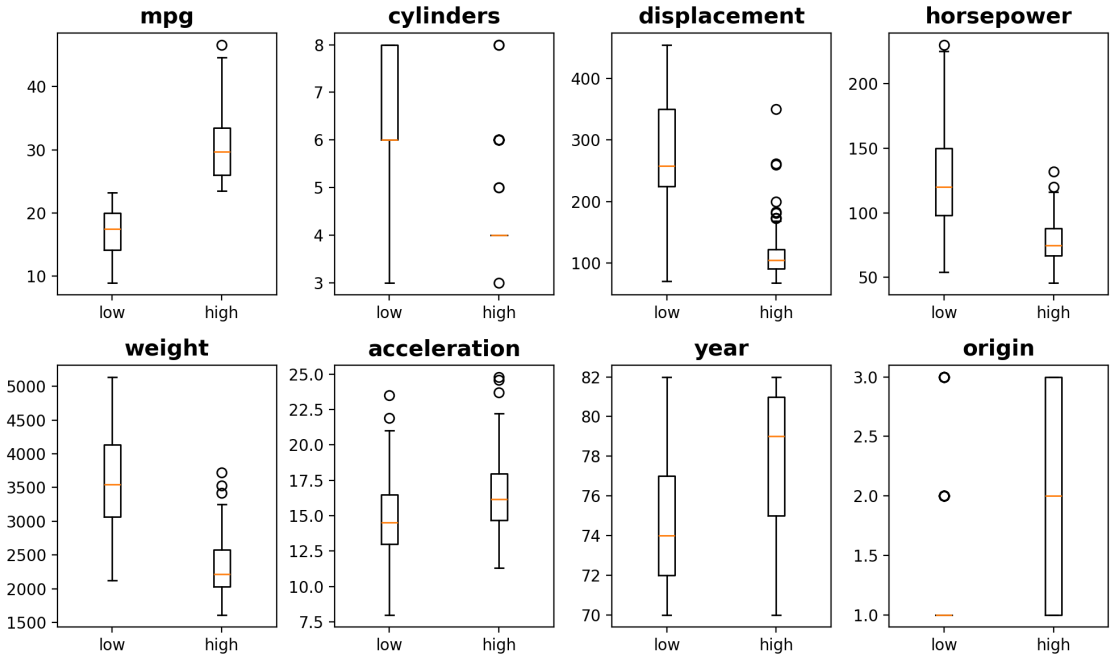
```
                mpg  cylinders  displacement  horsepower  weight  \
mpg           1.000     -0.778        -0.805      -0.778  -0.832
cylinders    -0.778      1.000         0.951       0.843   0.898
displacement -0.805      0.951         1.000       0.897   0.933
horsepower   -0.778      0.843         0.897       1.000   0.865
weight       -0.832      0.898         0.933       0.865   1.000
```

```
acceleration  0.423     -0.505        -0.544      -0.689  -0.417
year          0.581     -0.346        -0.370      -0.416  -0.309

              acceleration   year
mpg                  0.423  0.581
cylinders           -0.505 -0.346
displacement        -0.544 -0.370
horsepower          -0.689 -0.416
weight              -0.417 -0.309
acceleration         1.000  0.290
year                 0.290  1.000
```



[18]:
```python
### Box Plot
plt.figure(figsize = (10,6), dpi=200)
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.boxplot(Auto.iloc[(Auto.iloc[:,8] == 'low').values,i], positions=[0])
    plt.boxplot(Auto.iloc[(Auto.iloc[:,8] == 'high').values,i], positions=[1])
    plt.xticks(range(2),['low', 'high'])
    plt.title(Auto.columns[i], fontsize=14, fontweight='bold')
plt.tight_layout()
```

```
plt.show()
```


mpg
cylinders
displacement
horsepower
weight
acceleration
year
origin
low
high

2 Bayesian Classifier

Bayes' Theorem:

$$\mathrm{P}\left(C|X=\mathbf{x}\right)=\frac{\mathrm{P}\left(C\right)\ \mathrm{P}\left(X=\mathbf{x}|C\right)}{\mathrm{P}\left(X=\mathbf{x}\right)}$$

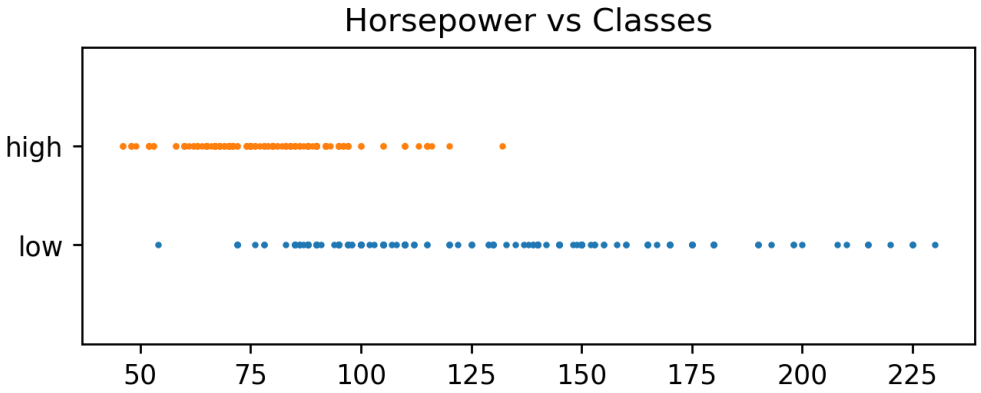
Notations:

- $\mathrm{P}\left(C|X=\mathbf{x}\right)$: Posterier
- $\mathrm{P}\left(C\right)$: Prior
- $\mathrm{P}\left(X=\mathbf{x}|C\right)$: Likelihood
- $\mathrm{P}\left(X=\mathbf{x}\right)$: Evidence

2.1 Considering Single Variate Classification

Using horsepower to determine whether the vehicle has high mpg or not.

[19]:
```
plt.figure(figsize = (6,2), dpi=250)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'low').values,3],
            Auto.iloc[(Auto.iloc[:,8] == 'low').values,8], s=2)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'high').values,3],
            Auto.iloc[(Auto.iloc[:,8] == 'high').values,8], s=2)
plt.ylim([-1,2])
plt.title("Horsepower vs Classes")
plt.show()
```



2.1.1 Hand-Crafting [1]: Using PDF

[20]:
```
# Calculate prior, mean, sd
hp_high = Auto.iloc[(Auto.iloc[:,8] == 'high').values,3]
hp_low = Auto.iloc[(Auto.iloc[:,8] == 'low').values,3]

mu_hp_high = np.mean(hp_high)
mu_hp_low = np.mean(hp_low)
```

```
sd_hp_high = np.std(hp_high)
sd_hp_low = np.std(hp_low)

# Prior
pr_high = len(hp_high) / (len(hp_high)+len(hp_low))
pr_low = len(hp_low) / (len(hp_high)+len(hp_low))
```

[21]:
```
# Define Normal Distribution PDF
def dnorm(x, mu, sd):
    exponent = np.exp(-(x - mu)**2 / (2 * sd**2))
    return 1/np.sqrt(2 * np.pi * sd**2) * exponent
```

Normal Distribution PDF:

$$f\left(x;\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{\left(x-\mu\right)^2}{2\sigma^2}\right]$$

[22]:
```
# Evaluate on new data
new_x = 90
# Evidence
evidence = dnorm(new_x, mu_hp_high, sd_hp_high) * pr_high + dnorm(new_x,␣
 ↪mu_hp_low, sd_hp_low) * pr_low
# Posterior
pr_x_high = pr_high * dnorm(new_x, mu_hp_high, sd_hp_high) / evidence
pr_x_low = pr_low * dnorm(new_x, mu_hp_low, sd_hp_low) / evidence
print("Probibilaty of new_x belongs to high mpg: ", round(pr_x_high,3))
print("Probibilaty of new_x belongs to low mpg: ", round(pr_x_low,3))
```

```
Probibilaty of new_x belongs to high mpg:  0.733
Probibilaty of new_x belongs to low mpg:  0.267
```

2.1.2 Hand-Crafting [2]: Using discriminant function

[23]:
```
# Calculate prior, mean, sd
hp_high = Auto.iloc[(Auto.iloc[:,8] == 'high').values,3]
hp_low = Auto.iloc[(Auto.iloc[:,8] == 'low').values,3]

mu_hp_high = np.mean(hp_high)
mu_hp_low = np.mean(hp_low)
sd_hp_high = np.std(hp_high)
sd_hp_low = np.std(hp_low)

# Prior
pr_high = len(hp_high) / (len(hp_high)+len(hp_low))
pr_low = len(hp_low) / (len(hp_high)+len(hp_low))
```

[24]:
```
def discri_fn(x, mu, sd, prior):
    g_x = -1 * np.log(sd)- (x - mu)**2 / (2 * sd**2) + np.log(prior)
```

```
    return g_x
```

[25]:
```
# Evaluate on new data
new_x = 90
pr_x_high = discri_fn(new_x, mu_hp_high, sd_hp_high, pr_high)
pr_x_low = discri_fn(new_x, mu_hp_low, sd_hp_low, pr_low)
```

[26]:
```
print("Discriminant function calculation of new_x belongs to high mpg: ",␣
 ↪round(pr_x_high,3))
print("Discriminant function calculation of new_x belongs to low mpg: ",␣
 ↪round(pr_x_low,3))
```

```
Discriminant function calculation of new_x belongs to high mpg:  -3.777
Discriminant function calculation of new_x belongs to low mpg:  -4.786
```

2.1.3 Done by scikit-learn package

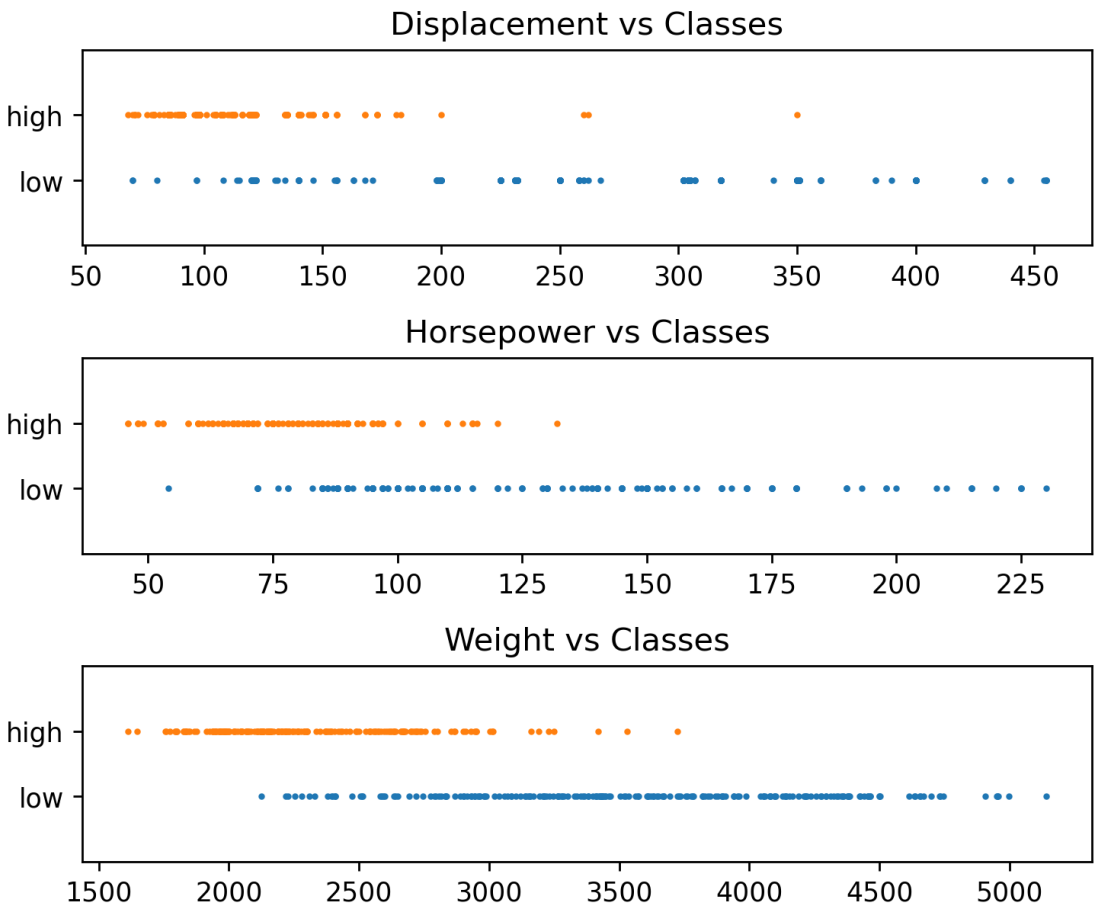
[27]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
model = gnb.fit(Auto.iloc[:,3].values.reshape(-1, 1), Auto.iloc[:,8].values)
y_pred = model.predict_proba(np.array(new_x).reshape(-1, 1))
print('[high, low] = ',np.round(y_pred,3))
```

```
[high, low] =  [[0.733 0.267]]
```

2.2 Considering Multi Variate Classification

Using displacement, horsepower, and weight to determine whether the vehicle has high mpg or not.

[28]:
```
plt.figure(figsize = (6,5), dpi=250)
plt.subplot(3,1,1)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'low').values,2],
            Auto.iloc[(Auto.iloc[:,8] == 'low').values,8], s=2)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'high').values,2],
            Auto.iloc[(Auto.iloc[:,8] == 'high').values,8], s=2)
plt.ylim([-1,2])
plt.title("Displacement vs Classes")

plt.subplot(3,1,2)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'low').values,3],
            Auto.iloc[(Auto.iloc[:,8] == 'low').values,8], s=2)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'high').values,3],
            Auto.iloc[(Auto.iloc[:,8] == 'high').values,8], s=2)
plt.ylim([-1,2])
plt.title("Horsepower vs Classes")

plt.subplot(3,1,3)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'low').values,4],
            Auto.iloc[(Auto.iloc[:,8] == 'low').values,8], s=2)
plt.scatter(Auto.iloc[(Auto.iloc[:,8] == 'high').values,4],
            Auto.iloc[(Auto.iloc[:,8] == 'high').values,8], s=2)
plt.ylim([-1,2])
plt.title("Weight vs Classes")
plt.tight_layout()
plt.show()
```



2.2.1 (a)-1 Naive Bayes Assumption [Hand Crafting]

[29]:
```
# Calculate prior, mean, sd
x_high = Auto.iloc[(Auto.iloc[:,8] == 'high').values,2:5]
x_low = Auto.iloc[(Auto.iloc[:,8] == 'low').values,2:5]

mu_high = np.mean(x_high, axis=0).values
mu_low = np.mean(x_low, axis=0).values
sd_high = np.std(x_high, axis=0).values
sd_low = np.std(x_low, axis=0).values

# Prior
pr_high = len(x_high) / (len(x_high)+len(x_low))
pr_low = len(x_low) / (len(x_high)+len(x_low))
```

[30]:
```
# Evaluate on new data
#new_x = np.array([100, 75, 2500])
new_x = np.array([150, 125, 2500])
```

```
# Evidence
evidence = (pr_high * (dnorm(new_x[0], mu_high[0], sd_high[0])
                       * dnorm(new_x[1], mu_high[1], sd_high[1])
                       * dnorm(new_x[2], mu_high[2], sd_high[2])) +
            pr_low * (dnorm(new_x[0], mu_low[0], sd_low[0])
                      * dnorm(new_x[1], mu_low[1], sd_low[1])
                      * dnorm(new_x[2], mu_low[2], sd_low[2])))

# Posterior
pr_x_high = pr_high * (dnorm(new_x[0], mu_high[0], sd_high[0])
                       * dnorm(new_x[1], mu_high[1], sd_high[1])
                       * dnorm(new_x[2], mu_high[2], sd_high[2])) / evidence
pr_x_low = pr_low * (dnorm(new_x[0], mu_low[0], sd_low[0])
                     * dnorm(new_x[1], mu_low[1], sd_low[1])
                     * dnorm(new_x[2], mu_low[2], sd_low[2])) / evidence
print("Probibilaty of new_x belongs to high mpg: ", round(pr_x_high,3))
print("Probibilaty of new_x belongs to low mpg: ", round(pr_x_low,3))
```

```
Probibilaty of new_x belongs to high mpg:  0.304
Probibilaty of new_x belongs to low mpg:  0.696
```

2.2.2 (a)-2 Naive Bayes Assumption [Package]

[31]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
model = gnb.fit(Auto.iloc[:,2:5].values, Auto.iloc[:,8].values)
y_pred = model.predict_proba(np.array(new_x).reshape(1,-1))
print('[high, low] = ',np.round(y_pred,3))
```

```
[high, low] =  [[0.304 0.696]]
```

2.2.3 (b)-1 Multivariate Assumption [Hand Crafting]

[32]:
```
# Calculate prior, mean, sd
x_high = Auto.iloc[(Auto.iloc[:,8] == 'high').values,2:5]
x_low = Auto.iloc[(Auto.iloc[:,8] == 'low').values,2:5]

mu_high = np.mean(x_high, axis=0)
mu_low = np.mean(x_low, axis=0)
cov_high = np.cov(x_high.T)
cov_low = np.cov(x_low.T)

# Prior
pr_high = len(x_high) / (len(x_high)+len(x_low))
pr_low = len(x_low) / (len(x_high)+len(x_low))
```

[33]:
```
# Define MultiNormal Distribution PDF
def dmultinorm(x, mu, cov):
    exponent = np.exp(-1/2*(x-mu).T.dot(np.linalg.inv(cov)).dot(x-mu))
    D = 1/np.sqrt((2 * np.pi)**len(x) * np.linalg.det(cov)) * exponent
    return D
```

Multivariate Normal Distribution PDF:

$$f(\mathbf{x};\ ,\ ) = \frac{1}{\sqrt{(2\pi)^d\ |\ |}} \exp\left[-\frac{1}{2}(\mathbf{x}-\ )^T\ ^{-1}(\mathbf{x}-\ )\right]$$

[34]:
```
# Evaluate on new data
#new_x = np.array([100, 75, 2500])
new_x = np.array([150, 125, 2500])

# Evidence
evidence = dmultinorm(new_x, mu_high, cov_high) * pr_high + dmultinorm(new_x,␣
 ↪mu_low, cov_low) * pr_low
# Posterior
pr_x_high = pr_high * dmultinorm(new_x, mu_high, cov_high) / evidence
pr_x_low = pr_low * dmultinorm(new_x, mu_low, cov_low) / evidence
print("Probibilaty of new_x belongs to high mpg: ", round(pr_x_high,3))
print("Probibilaty of new_x belongs to low mpg: ", round(pr_x_low,3))
```

```
Probibilaty of new_x belongs to high mpg:  0.257
Probibilaty of new_x belongs to low mpg:  0.743
```

3 Implement on mpg Dataset

3.1 Train Test Split

Split Dataset into 70% training set & 30% test set randomly.

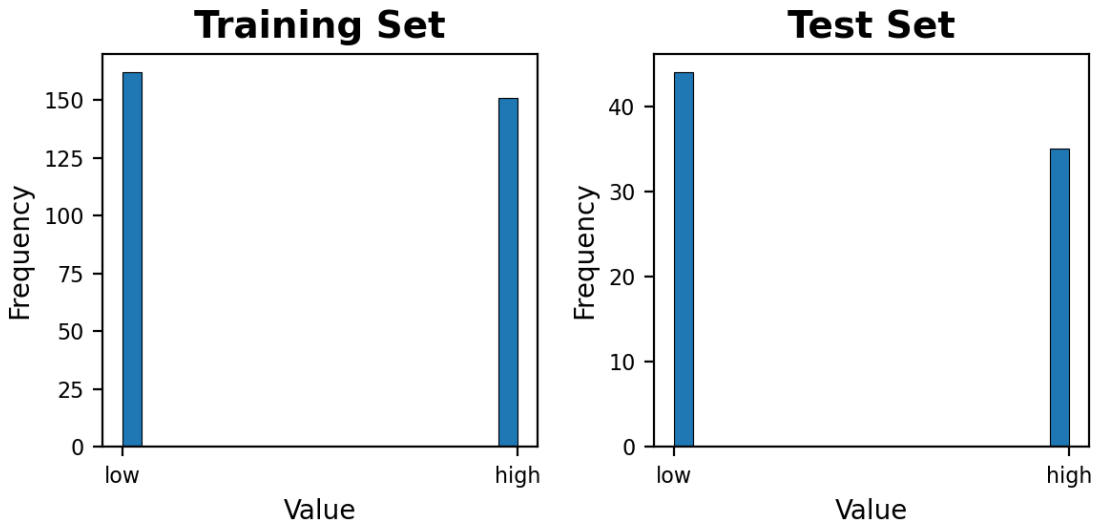
[35]:
```
np.random.seed(524773)
train_num = np.sort(np.random.choice(np.arange(len(Auto)), int(len(Auto)*0.8),␣
 ↪replace=False))
test_num = np.delete(np.arange(len(Auto)), train_num)
```

[36]:
```
print("Training number: \n", train_num)
print("Test number: \n", test_num)
```

```
Training number:
 [  1   3   4   5   6   7   8  11  13  14  15  16  17  18  19  20  21  22
  23  24  25  27  29  30  31  32  33  34  35  37  40  41  42  44  45  46
  47  48  50  52  53  54  56  57  58  59  60  61  62  63  64  65  66  67
  70  71  72  73  74  76  77  78  79  80  81  82  83  84  85  86  87  89
  90  92  93  94  96  97  99 100 102 103 104 105 106 107 108 109 110 111
 112 113 115 116 118 119 120 121 122 123 125 126 127 128 129 130 131 132
 134 135 136 137 138 139 142 144 145 146 147 148 150 151 152 153 154 155
 156 157 158 159 160 161 162 163 164 166 168 169 170 171 172 173 174 175
 176 180 181 182 183 185 186 187 188 189 190 191 192 193 194 195 196 197
 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
 217 218 221 222 223 225 226 228 229 232 233 234 235 236 237 238 240 242
 245 246 249 252 253 255 256 257 259 260 261 262 263 264 265 266 267 268
 269 270 271 274 275 276 277 280 281 282 284 285 286 287 289 290 291 292
 293 295 296 297 299 301 302 303 304 305 306 308 309 310 313 314 315 316
 317 318 319 321 322 323 324 325 326 328 329 331 336 337 338 339 341 342
 343 344 346 348 349 350 351 352 353 354 355 357 359 360 361 362 363 364
 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 381 382 383
 384 385 387 388 389 390 391]
Test number:
 [  0   2   9  10  12  26  28  36  38  39  43  49  51  55  68  69  75  88
  91  95  98 101 114 117 124 133 140 141 143 149 165 167 177 178 179 184
 198 219 220 224 227 230 231 239 241 243 244 247 248 250 251 254 258 272
 273 278 279 283 288 294 298 300 307 311 312 320 327 330 332 333 334 335
 340 345 347 356 358 380 386]
```

[37]:
```
plt.figure(figsize = (6,3), dpi=200)
plt.subplot(1,2,1)
plt.hist(Auto.iloc[train_num,8], bins=20, rwidth=1, edgecolor='black',␣
 ↪linewidth=0.4)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.xlabel('Value', fontsize=10)
plt.ylabel('Frequency', fontsize=10)
```

```
plt.title("Training Set", fontsize=14, fontweight='bold')

plt.subplot(1,2,2)
plt.hist(Auto.iloc[test_num,8], bins=20, rwidth=1, edgecolor='black',␣
 ↪linewidth=0.4)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.xlabel('Value', fontsize=10)
plt.ylabel('Frequency', fontsize=10)
plt.title("Test Set", fontsize=14, fontweight='bold')

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



3.2 Naive Bayes: Hand Crafting

[38]:
```
Auto_train = Auto.iloc[train_num,:]
Auto_test = Auto.iloc[test_num,:]
print("Training data:\n")
display(Auto_train)
print("Test data:\n")
display(Auto_test)
```

```
Training data:


      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
1    15.0          8         350.0       165.0    3693          11.5    70
3    16.0          8         304.0       150.0    3433          12.0    70
4    17.0          8         302.0       140.0    3449          10.5    70
5    15.0          8         429.0       198.0    4341          10.0    70
6    14.0          8         454.0       220.0    4354           9.0    70
..    …         …           …          …      …           …   …
392  27.0          4         140.0        86.0    2790          15.6    82
393  44.0          4          97.0        52.0    2130          24.6    82
394  32.0          4         135.0        84.0    2295          11.6    82
395  28.0          4         120.0        79.0    2625          18.6    82
396  31.0          4         119.0        82.0    2720          19.4    82

     origin mpg_label  mpg_label_01
1         1       low             1
3         1       low             1
4         1       low             1
5         1       low             1
6         1       low             1
..      …       …           …
392       1      high             0
393       2      high             0
394       1      high             0
395       1      high             0
396       1      high             0

[313 rows x 10 columns]

Test data:


      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0       130.0    3504          12.0    70
2    18.0          8         318.0       150.0    3436          11.0    70
9    15.0          8         390.0       190.0    3850           8.5    70
10   15.0          8         383.0       170.0    3563          10.0    70
12   15.0          8         400.0       150.0    3761           9.5    70
..    …         …           …          …      …           …   …
```

```
351  34.4          4          98.0        65.0    2045          16.2    81
361  25.4          6         168.0       116.0    2900          12.6    81
363  22.4          6         231.0       110.0    3415          15.8    81
385  25.0          6         181.0       110.0    2945          16.4    82
391  27.0          4         151.0        90.0    2950          17.3    82

     origin mpg_label  mpg_label_01
0         1       low             1
2         1       low             1
9         1       low             1
10        1       low             1
12        1       low             1
..      …       …           …
351       1      high             0
361       3      high             0
363       1       low             1
385       1      high             0
391       1      high             0

[79 rows x 10 columns]
```

Selected features for classification: - cylinders - displacement - horsepower - weight - acceleration

[39]:
```
select_feature = ["cylinders", "displacement", "horsepower", "weight",␣
 ↪"acceleration", "mpg_label"]

mu = Auto_train[select_feature].groupby("mpg_label").mean()
sd = Auto_train[select_feature].groupby("mpg_label").std()
count = Auto_train[select_feature].groupby("mpg_label").count().iloc[:,0].
 ↪rename('counts')
```

[40]:
```
print("Mean:\n")
display(mu)
print("Standard deivation:\n")
display(sd)
print("Counts:\n")
display(count)
```

```
Mean:


           cylinders  displacement  horsepower       weight  acceleration
mpg_label
high        4.125828    113.069536   78.099338  2315.927152     16.550331
low         6.592593    265.141975  127.098765  3555.469136     14.742593

Standard deivation:
```

```
           cylinders  displacement  horsepower      weight  acceleration
mpg_label
high        0.545339     34.037187   14.907607  376.076909      2.529397
low         1.522321     93.563468   38.653294  722.788314      2.711645
```

```
Counts:


mpg_label
high    151
low     162
Name: counts, dtype: int64
```

[41]:
```
prior = count / count.sum()
```

[42]:
```
pr_high_all = []
pr_low_all = []
pred_label = []
for num in range(Auto_test.shape[0]):
    data = Auto_test[select_feature].iloc[num,:].drop('mpg_label')
    pr_high = 1
    pr_low = 1
    for col in range(data.shape[0]):
        pr_high = pr_high * dnorm(data.iloc[col], mu.loc['high'].iloc[col], sd.
 ↪loc['high'].iloc[col])
        pr_low = pr_low * dnorm(data.iloc[col], mu.loc['low'].iloc[col], sd.
 ↪loc['low'].iloc[col])
    pr_high = pr_high * prior['high']
    pr_low = pr_low * prior['low']
    evidence = pr_high + pr_low
    pr_high = pr_high / evidence
    pr_low = pr_low / evidence
    pr_high_all.append(pr_high)
    pr_low_all.append(pr_low)
    if pr_high > pr_low:
        pred_label.append('high')
    else:
        pred_label.append('low')
```

[43]:
```
Auto_test.loc[:,'Prob. High'] = pr_high_all
Auto_test.loc[:,'Prob. Low'] = pr_low_all
Auto_test.loc[:,'Pred. Label'] = pred_label
```

```
/var/folders/k2/gx9q4khd66vfq69v2qr3drvh0000gn/T/ipykernel_4356/2843818983.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Auto_test.loc[:,'Prob. High'] = pr_high_all
/var/folders/k2/gx9q4khd66vfq69v2qr3drvh0000gn/T/ipykernel_4356/2843818983.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Auto_test.loc[:,'Prob. Low'] = pr_low_all
/var/folders/k2/gx9q4khd66vfq69v2qr3drvh0000gn/T/ipykernel_4356/2843818983.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Auto_test.loc[:,'Pred. Label'] = pred_label
```

[44]:
```
from sklearn.metrics import confusion_matrix
accuracy = np.sum((Auto_test['mpg_label'] == pred_label))/len(Auto_test)
conf_mat = confusion_matrix(Auto_test['mpg_label'], pred_label, labels=["high",␣
 ↪"low"])
print("Accuracy: ", round(accuracy,3))
print("Confusion matrix: \n", conf_mat)
```

```
Accuracy:  0.899
Confusion matrix:
 [[31  4]
 [ 4 40]]
```

[45]:
```
Auto_test[select_feature].groupby("mpg_label").count().iloc[:,0].
 ↪rename('counts')
```

[45]:
```
mpg_label
high    35
low     44
Name: counts, dtype: int64
```

3.3 Naive Bayes: Package

[46]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
model = gnb.fit(Auto_train[select_feature].drop(columns = ['mpg_label']),␣
 ↪Auto_train['mpg_label'])
pred_prob = model.predict_proba(Auto_test[select_feature].drop(columns =␣
 ↪['mpg_label']))
pred_label = model.predict(Auto_test[select_feature].drop(columns =␣
 ↪['mpg_label']))
```

[47]:
```
accuracy = np.sum((Auto_test['mpg_label'] == pred_label))/len(Auto_test)
conf_mat = confusion_matrix(Auto_test['mpg_label'], pred_label, labels=["high",␣
 ↪"low"])
print("Accuracy: ", round(accuracy,3))
print("Confusion matrix: \n", conf_mat)
```

```
Accuracy:  0.899
Confusion matrix:
 [[31  4]
 [ 4 40]]
```

[48]:
```
Auto_test[select_feature].groupby("mpg_label").count().iloc[:,0].
 ↪rename('counts')
```

[48]:
```
mpg_label
high    35
low     44
Name: counts, dtype: int64
```

[ ]:

[ ]: