In [1]: import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

    import sklearn.preprocessing as skl_pre
    import sklearn.linear_model as skl_lm
    import sklearn.discriminant_analysis as skl_da
    import sklearn.neighbors as skl_nb

    plt.style.use('seaborn-white')

1  4.1 Getting started with classification – Breast cancer diagnosis

In this exercise, we will consider the data set Data/biopsy.csv with data from breast biopsies, for the purpose of diagnosing breast cancer. For each patient, the data set contains nine different attributes (clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli and mitoses) scored on a scale from 1 to 10, as well as the physician’s diagnosis (malign or benign).

1.1 Dataset

This data frame biopsy contains the following columns:
ID: sample code number (not unique).
V1: clump thickness.
V2: uniformity of cell size.
V3: uniformity of cell shape.
V4: marginal adhesion.
V5: single epithelial cell size.
V6: bare nuclei (16 values are missing).
V7: bland chromatin.
V8: normal nucleoli.
V9: mitoses.
class: "benign" or "malignant".
1.2  a)

Load and familiarize yourself with the data set, using, e.g. info(), describe(), pandas.plotting.scatter_matrix() and print().

In 

```python
np.random.seed(1)
biopsy = pd.read_csv('Data/biopsy.csv', na_values='?', dtype={'ID': str}).dropna().reset_index()
```

```python
print(biopsy)
# Pretty alternative: display(HTML(biopsy.head().to_html())),
# requires "from IPython.display import display, HTML"
biopsy.info()
print(biopsy.columns.values)
```

<table>
<thead>
<tr>
<th></th>
<th>ID</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
<th>V6</th>
<th>V7</th>
<th>V8</th>
<th>V9</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1000025</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1002945</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>10.0</td>
<td>3</td>
<td>2</td>
<td>benign</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1015425</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1016277</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>4.0</td>
<td>3</td>
<td>7</td>
<td>benign</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1017023</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1017122</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>10.0</td>
<td>9</td>
<td>7</td>
<td>1 malignant</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1018099</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>10.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1018561</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1033078</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>benign</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1033078</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1035283</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1036172</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>1041801</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3.0</td>
<td>4</td>
<td>4</td>
<td>1 malignant</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>1043999</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>1044572</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td>9.0</td>
<td>5</td>
<td>5</td>
<td>4 malignant</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>1047630</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>1.0</td>
<td>4</td>
<td>3</td>
<td>1 malignant</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>1048672</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>1049815</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>1050670</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>10.0</td>
<td>4</td>
<td>1</td>
<td>2 malignant</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>1050718</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>1054590</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>5</td>
<td>10.0</td>
<td>5</td>
<td>4</td>
<td>4 malignant</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>1054593</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7.0</td>
<td>7</td>
<td>10</td>
<td>1 malignant</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>1056784</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>1059552</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>1065726</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7.0</td>
<td>3</td>
<td>6</td>
<td>1 malignant</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>1066373</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>1066979</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>1067444</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>1070935</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>1070935</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>653</td>
<td>653</td>
<td>1350423</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>5.0</td>
<td>7</td>
<td>10</td>
<td>1 malignant</td>
</tr>
<tr>
<td>654</td>
<td>670</td>
<td>1352848</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>8.0</td>
<td>7</td>
<td>4</td>
<td>1 malignant</td>
</tr>
<tr>
<td>655</td>
<td>671</td>
<td>1353092</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>benign</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>V1</td>
<td>V2</td>
<td>V3</td>
<td>V4</td>
<td>V5</td>
<td>V6</td>
<td>V7</td>
<td>V8</td>
<td>V9</td>
<td>class</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>656</td>
<td>672</td>
<td>1354840</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>657</td>
<td>673</td>
<td>1354840</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>658</td>
<td>674</td>
<td>1355260</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>659</td>
<td>675</td>
<td>1365075</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>660</td>
<td>676</td>
<td>1365328</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>661</td>
<td>677</td>
<td>1368267</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>662</td>
<td>678</td>
<td>1368273</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>663</td>
<td>679</td>
<td>1368882</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>664</td>
<td>680</td>
<td>1369821</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>10.0</td>
<td>10</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>665</td>
<td>681</td>
<td>1371026</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>4</td>
<td>10.0</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>666</td>
<td>682</td>
<td>1371920</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>667</td>
<td>683</td>
<td>466906</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>668</td>
<td>684</td>
<td>466906</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>669</td>
<td>685</td>
<td>534555</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>670</td>
<td>686</td>
<td>536708</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>671</td>
<td>687</td>
<td>566346</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>672</td>
<td>688</td>
<td>603148</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>673</td>
<td>689</td>
<td>654546</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>674</td>
<td>690</td>
<td>654546</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>675</td>
<td>691</td>
<td>695091</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>5.0</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>676</td>
<td>692</td>
<td>714039</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>677</td>
<td>693</td>
<td>763235</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>678</td>
<td>694</td>
<td>776715</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>679</td>
<td>695</td>
<td>841769</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>680</td>
<td>696</td>
<td>888820</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>3.0</td>
<td>8</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>681</td>
<td>697</td>
<td>897471</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>4.0</td>
<td>10</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>682</td>
<td>698</td>
<td>897471</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>5.0</td>
<td>10</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

[683 rows x 12 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 683 entries, 0 to 682
Data columns (total 12 columns):
index   683 non-null int64
ID      683 non-null object
V1      683 non-null int64
V2      683 non-null int64
V3      683 non-null int64
V4      683 non-null int64
V5      683 non-null int64
V6      683 non-null float64
V7      683 non-null int64
V8      683 non-null int64
V9      683 non-null int64
class   683 non-null object
dtypes: float64(1), int64(9), object(2)
memory usage: 64.1+ KB
['index' 'ID' 'V1' 'V2' 'V3' 'V4' 'V5' 'V6' 'V7' 'V8' 'V9' 'class']
In [3]: # Summary of the variables, note that it does not include the string variables
biopsy.describe()

Out[3]:

<table>
<thead>
<tr>
<th></th>
<th>index</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>683.000000</td>
<td>683.000000</td>
<td>683.000000</td>
<td>683.000000</td>
<td>683.000000</td>
<td>683.000000</td>
</tr>
<tr>
<td>mean</td>
<td>351.355783</td>
<td>4.442167</td>
<td>3.15085</td>
<td>3.215227</td>
<td>2.830161</td>
<td>3.234261</td>
</tr>
<tr>
<td>std</td>
<td>202.563927</td>
<td>2.820761</td>
<td>3.065145</td>
<td>2.988581</td>
<td>2.864562</td>
<td>2.223085</td>
</tr>
<tr>
<td>min</td>
<td>0.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>25%</td>
<td>176.500000</td>
<td>2.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>50%</td>
<td>355.000000</td>
<td>4.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>75%</td>
<td>526.500000</td>
<td>6.000000</td>
<td>5.000000</td>
<td>5.000000</td>
<td>4.000000</td>
<td>4.000000</td>
</tr>
<tr>
<td>max</td>
<td>698.000000</td>
<td>10.000000</td>
<td>10.000000</td>
<td>10.000000</td>
<td>10.000000</td>
<td>10.000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>V6</th>
<th>V7</th>
<th>V8</th>
<th>V9</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>683.000000</td>
<td>683.000000</td>
<td>683.000000</td>
<td>683.000000</td>
</tr>
<tr>
<td>mean</td>
<td>3.544656</td>
<td>3.445095</td>
<td>2.869693</td>
<td>1.603221</td>
</tr>
<tr>
<td>std</td>
<td>3.643857</td>
<td>2.449697</td>
<td>3.052666</td>
<td>1.732674</td>
</tr>
<tr>
<td>min</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>25%</td>
<td>1.000000</td>
<td>2.000000</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>50%</td>
<td>1.000000</td>
<td>3.000000</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>75%</td>
<td>6.000000</td>
<td>5.000000</td>
<td>4.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>max</td>
<td>10.000000</td>
<td>10.000000</td>
<td>10.000000</td>
<td>10.000000</td>
</tr>
</tbody>
</table>

In [4]: # scatterplot of the variables V1-V9
pd.plotting.scatter_matrix(biopsy.iloc[:, 2:11])
plt.show()
### 1.3 b)

Split the data randomly into a training set and a test set of approximately similar size.

```
In [5]: # sampling indices for training
    trainI = np.random.choice(biopsy.shape[0], size=300, replace=False)
    trainIndex = biopsy.index.isin(trainI)
    train = biopsy.iloc[trainIndex]  # training set
    test = biopsy.iloc[-trainIndex]  # test set
```

### 1.4 c) Logistic regression

Perform logistic regression with class as output variable and V3, V4 and V5 as input variables. Do a prediction on the test set, and compute (i) the fraction of correct predictions and (ii) the confusion matrix (using, for example, pandas.crosstab()). The commands `skl_lm.LogisticRegression()` and `model.predict()` are useful. Is the performance any good, and what does the confusion matrix tell you?

```
In [6]: model = skl_lm.LogisticRegression()
    X_train = train[['V3', 'V4', 'V5']]
    Y_train = train['class']
    X_test = test[['V3', 'V4', 'V5']]
    Y_test = test['class']

    model.fit(X_train, Y_train)
    print('Model summary: ')
    print(model)

    Model summary:
    LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                       intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                       penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                       verbose=0, warm_start=False)

In [7]: predict_prob = model.predict_proba(X_test)
    print('The class order in the model: ')
    print(model.classes_)
    print('Examples of predicted probabilities for the above classes: ')
    predict_prob[0:5]  # inspect the first 5 predictions

    The class order in the model:
    ['benign' 'malignant']
    Examples of predicted probabilities for the above classes:

    Out[7]: array([[0.07182943, 0.92817057],
                   [0.95882118, 0.04117882],
                   [0.78213423, 0.21786578],
                   [0.51863398, 0.48136602],
                   [0.98420384, 0.01579616]])
In [8]: prediction = np.empty(len(X_test), dtype=object)
   prediction = np.where(predict_prob[:, 0]>=0.5, 'benign', 'malignant')
   prediction[0:5]  # Inspect the first 5 predictions after labeling.

Out[8]: array(['malignant', 'benign', 'benign', 'benign', 'benign'], dtype='<U9')

In [9]: # Confusion matrix
   print(pd.crosstab(prediction, Y_test))
   # Accuracy
   np.mean(prediction == Y_test)

<table>
<thead>
<tr>
<th>class</th>
<th>benign</th>
<th>malignant</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>benign</td>
<td>239</td>
<td>13</td>
</tr>
<tr>
<td>malignant</td>
<td>11</td>
<td>120</td>
</tr>
</tbody>
</table>

Out[9]: 0.9373368146214099

1.5 d) LDA

Repeat(c) using LDA. A useful command is sklearn.discriminant_analysis.LinearDiscriminantAnalysis()
sklearn.discriminant_analysis is imported as skl_da

In [10]: model = skl_da.LinearDiscriminantAnalysis()
   model.fit(X_train, Y_train)

Out[10]: LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                                          solver='svd', store_covariance=False, tol=0.0001)

In [11]: predict_prob = model.predict_proba(X_test)
   print('The class order in the model: ')
   print(model.classes_)
   print('Examples of predicted probabilities for the above classes: ')
   predict_prob[0:5]  # inspect the first 5 predictions

The class order in the model:
['benign' 'malignant']
Examples of predicted probabilities for the above classes:

Out[11]: array([[7.83143570e-02, 9.21685643e-01],
                  [9.99486179e-01, 5.13821412e-04],
                  [9.99486179e-01, 5.13821412e-04],
                  [9.98159737e-01, 1.84026306e-03],
                  [9.99486179e-01, 5.13821412e-04]])
In [12]: prediction = np.empty(len(X_test), dtype=object)
prediction = np.where(predict_prob[:, 0] >= 0.5, 'benign', 'malignant')
print(prediction[0:5])  # Inspect the first 5 predictions after labeling.

# Confusion matrix
print(pd.crosstab(prediction, Y_test))

# Accuracy
np.mean(prediction == Y_test)

Out[12]: 0.9164490861618799

1.6 e) QDA
Repeat (c) using QDA. A useful command is sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis().

In [13]: model = skl.da.QuadraticDiscriminantAnalysis()
   : model.fit(X_train, Y_train)

Out[13]: QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
   : store_covariance=False, store_covariances=None, tol=0.0001)

In [14]: predict_prob = model.predict_proba(X_test)
   : print('The class order in the model:')
   : print(model.classes_)
   : print('Examples of predicted probabilities for the above classes:')
   : predict_prob[0:5]  # inspect the first 5 predictions

The class order in the model:
['benign' 'malignant']
Examples of predicted probabilities for the above classes:

Out[14]: array([[1.08710486e-10, 1.00000000e+00],
   : [9.99184518e-01, 8.15482107e-04],
   : [9.99184518e-01, 8.15482107e-04],
   : [9.98215859e-01, 1.78414072e-03],
   : [9.99184518e-01, 8.15482107e-04]])

In [15]: prediction = np.empty(len(X_test), dtype=object)
prediction = np.where(predict_prob[:, 0] >= 0.5, 'benign', 'malignant')
print(prediction[0:5])  # Inspect the first 5 predictions after labeling.

# Confusion matrix
print(pd.crosstab(prediction, Y_test))

# Accuracy
np.mean(prediction == Y_test)

[['malignant' 'benign' 'benign' 'benign' 'benign']
class benign malignant
row_0 benign 237 12
malignant 13 121

Out[15]: 0.9347258485639687

1.7  f) KNN

Repeat (c) using k-NN (with \( k = 1 \)). A useful commands is sklearn.neighbors.KNeighborsClassifier().

In [16]: model = skl_nb.KNeighborsClassifier(n_neighbors=1)
model.fit(X_train, Y_train)

Out[16]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=1, p=2,
weights='uniform')

In [17]: prediction = model.predict(X_test)
print(pd.crosstab(prediction, Y_test))
np.mean(prediction == Y_test)

class benign malignant
row_0 benign 238 19
malignant 12 114

Out[17]: 0.9190600522193212

1.8  g) Try different values for KNN

Use a for-loop to explore the performance of k-NN for different values of \( k \), and plot the fraction of correct predictions as a function of \( k \).

In [18]: misclassification = []
for k in range(50):  # Try n_neighbors = 1, 2, \ldots, 50
    model = skl_nb.KNeighborsClassifier(n_neighbors=k+1)
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
misclassification.append(np.mean(prediction != Y_test))

1.9  h) ROC for logistic regression

Use a for-loop to explore how the true and false positive rates in logistic regression are affected by different threshold values, and plot the result as a ROC curve. (see Figure 4.8 and Table 4.6 and 4.7 in ISL).

In [19]:
false_positive_rate = []
true_positive_rate = []

N = np.sum(Y_test == 'benign')
P = np.sum(Y_test == 'malignant')

threshold = np.linspace(0.01, 0.99, 99)
model = skl_lm.LogisticRegression(solver='liblinear')
model.fit(X_test, Y_test)
predict_prob = model.predict_proba(X_test)

for i in range(len(threshold)):
    prediction = np.empty(len(X_test), dtype=object)
    prediction = np.where(predict_prob[:, 0] > threshold[i],
                          'benign', 'malignant')
FP = np.sum((prediction=='malignant')&(Y_test=='benign'))
TP = np.sum((prediction=='malignant')&(Y_test=='malignant'))

false_positive_rate.append(FP/N)
true_positive_rate.append(TP/P)

plt.plot(false_positive_rate, true_positive_rate)

Out[19]: [<matplotlib.lines.Line2D at 0x16bfb7387f0>]

1.10  i)
Try to find another set of inputs (perhaps by also considering transformations of the attributes) which gives a better result than you have achieved so far. You may also play with the threshold values. ("Better" is on purpose left vague. For this problem, the implications of a false negative (=benign) misclassification is probably more severe than a false positive (=malignant) misclassification.)

2  4.2 Decision boundaries
The following code generates some data with $x_1$ and $x_2$ both in $[0, 10]$ and $y$ either 0 or 1, and plots the decision boundary for a logistic regression model.

In [20]: # generate data
np.random.seed(2)
N = 100
x1 = np.random.uniform(0, 10, N)
x2 = np.random.uniform(0, 10, N)
y = np.repeat(1, N)
y[x1<4] = 0
y[x2<4] = 0
X = pd.DataFrame({'x1': x1, 'x2': x2})

# learn a logistic regression model
model = skl_lm.LogisticRegression(solver='liblinear')
model.fit(X, y)

# classify the points in the whole domain
res = 0.1  # resolution of the squares
xs1 = np.arange(0, 10.1, 0.1)
xs2 = np.arange(0, 10.1, 0.1)
xs1, xs2 = np.meshgrid(xs1, xs2)  # Creating the grid for all the data points
X_all = pd.DataFrame({'x1': xs1.flatten(), 'x2': xs2.flatten()})
prediction = model.predict(X_all)

plt.figure(figsize=(10, 5))

# Plot of the prediction for all the points in the space
colors = np.where(prediction==0, 'skyblue', 'lightsalmon')
plt.scatter(xs1, xs2, s=90, marker='s', c=colors)

# Plot of the data points and their label
color = np.where(y==0, 'b', 'r')
plt.scatter(x1, x2, c=color)

plt.title('Logistic regression decision boundary')
plt.show()
2.1 (a)

Run the code and verify that it reproduces the figure, and make sure you understand the figure. What is the misclassification rate here?

In [21]: # In this problem, the misclassification rate for the logistic regression is 10%
  # (the number of points that are in the wrong region in the figure)

2.2 (b)

Modify the code to plot the decision boundary for a LDA classifier. What differences do you see? What is the misclassification rate?

In [22]: # learn a LDA model
    model = skl_da.LinearDiscriminantAnalysis()
    model.fit(X, y)

    # classify many points, and plot a colored square around each point
    res = 0.1    # resolution of the squares
    xs1 = np.arange(0, 10.1, 0.1)
    xs2 = np.arange(0, 10.1, 0.1)
    xs1, xs2 = np.meshgrid(xs1, xs2)  # Creating the grid for all the data points
    X_all = pd.DataFrame({'x1': xs1.flatten(), 'x2': xs2.flatten()})
    prediction = model.predict(X_all)

    plt.figure(figsize=(10, 5))

    # Plot of the prediction for all the points in the space
colors = np.where(prediction==0,'skyblue','lightsalmon')
plt.scatter(xs1, xs2, s = 90, marker='s', c=colors)

# Plot of the data points and their label
color = np.where(y==0, 'b', 'r')
plt.scatter(x1, x2, c=color)

plt.title('LDA decision boundary')
plt.show()

# Misclassification rate 10%.
# Note that the decision boundaries for both logistic regression and
# LDA are linear, but not identical.

2.3 (c)

Modify the code to plot the decision boundary for a QDA classifier. What differences do you see? What is the misclassification rate?

In [23]: # learn a QDA model
    model = skl_da.QuadraticDiscriminantAnalysis()
    model.fit(X, y)

    # classify many points, and plot a colored square around each point
    res = 0.1    # resolution of the squares
    xs1 = np.arange(0, 10.1, 0.1)
    xs2 = np.arange(0, 10.1, 0.1)
    xs1, xs2 = np.meshgrid(xs1, xs2)    # Creating the grid for all the data points

13
\[ X_{\text{all}} = \text{pd.DataFrame}({'x1': xs1.flatten(), 'x2': xs2.flatten()}) \]
\[ \text{prediction} = \text{model.predict}(X_{\text{all}}) \]

\[
\text{plt.figure(figsize=(10, 5))}
\]

\# Plot of the prediction for all the points in the space
\[ \text{colors} = \text{np.where}((\text{prediction}==0, 'skyblue', 'lightsalmon')) \]
\[ \text{plt.scatter}(x1, \text{x2}, s = 90, \text{marker}='s', \text{c}=\text{colors}) \]

\# Plot of the data points and their label
\[ \text{color} = \text{np.where}(y==0, 'b', 'r') \]
\[ \text{plt.scatter}(x1, \text{x2}, \text{c}=\text{color}) \]

\[ \text{plt.title('QDA decision boundary')} \]
\[ \text{plt.show()} \]

\# Misclassification rate 9%. The decision boundary of QDA is not linear.

2.4 (d)
Modify the code to plot the decision boundary for a k-NN classifier. What differences do you see? What is the misclassification rate?

In [24]: # learn a KNN model with k=1
\[
\text{model} = \text{skl_nb.KNeighborsClassifier(n_neighbors=1)}
\]
\[ \text{model.fit}(X, y) \]

\# classify many points, and plot a colored square around each point
res = 0.1  # resolution of the squares
xs1 = np.arange(0, 10.1, 0.1)
xs2 = np.arange(0, 10.1, 0.1)
xs1, xs2 = np.meshgrid(xs1, xs2)  # Creating the grid for all the data points
X_all = pd.DataFrame({'x1': xs1.flatten(), 'x2': xs2.flatten()})
prediction = model.predict(X_all)

plt.figure(figsize=(10, 5))

# Plot of the prediction for all the points in the space
colors = np.where(prediction==0, 'skyblue', 'lightsalmon')
plt.scatter(xs1, xs2, s=90, marker='s', c=colors)

# Plot of the data points and their label
color = np.where(y==0, 'b', 'r')
plt.scatter(x1, x2, c=color)

plt.title('KNN decision boundary')
plt.show()

# The misclassification rate is 0% (which always is the case when k = 1).
# The misclassification rate for a test data set could still be much worse.

2.5 (e)

What happens with the decision boundary for logistic regression if you include the term $x_1x_2$ as an input? What is the misclassification rate?
In [25]: # learn a logistic regression model including the term X1*X2 as an input

    X['x1x2'] = X['x1']*X['x2']
    model = skl_lm.LogisticRegression(solver='liblinear')
    model.fit(X, y)

    # classify many points, and plot a colored square around each point
    res = 0.1    # resolution of the squares
    xs1 = np.arange(0, 10.1, 0.1)
    xs2 = np.arange(0, 10.1, 0.1)
    xs1, xs2 = np.meshgrid(xs1, xs2)    # Creating the grid for all the data points
    X_all = pd.DataFrame({"x1": xs1.flatten(), "x2": xs2.flatten()})
    X_all['x1x2'] = X_all['x1'] * X_all['x2']
    prediction = model.predict(X_all)

    plt.figure(figsize=(10, 5))

    # Plot of the prediction for all the points in the space
    colors = np.where(prediction==0, 'skyblue', 'lightsalmon')
    plt.scatter(xs1, xs2, s=90, marker='s', c=colors)

    # Plot of the data points and their label
    color = np.where(y==0, 'b', 'r')
    plt.scatter(x1, x2, c=color)

    plt.title('KNN decision boundary')
    plt.show()

    # Misclassification rate 4%. Using nonlinear transformations of the inputs
    # is one way to create a nonlinear decision boundary in a linear model.
    # However, the decision boundary in a 3D-plot plot with axes
    # 'x1', 'x2' and 'x1x2' would still be linear.
4.3 Why not linear regression?

In this exercise, we explore why linear regression might not be well suited for classification problems.

3.1 (a)

Construct and plot a data set as follows: Let \( x_i \) be samples \( x_i = i \) in a sequence from \( i = 1 \) to \( i = 40 \). Let \( y_i = 0 \) for all \( i = 1 : 40 \), except for \( i = 34, 38, 39, 40 \) where \( y_i = 1 \). Hence, \( y \) belongs to either of two classes, 0 and 1.

\[
\begin{align*}
\text{In [26]: } & \quad x = \text{np.arange}(40)+1 \\
& \quad y = \text{np.repeat}(0, 40) \\
& \quad y[[33, 37, 38, 39]] = 1
\end{align*}
\]

3.2 (b)

Now, the problem is to fit a model which is able to predict the output \( y \) from the input \( x \). Start with a linear regression model (command \texttt{skl\_lm.LinearRegression()}) and simply threshold its predictions at 0.5 (the average of 0 and 1, the two classes). Plot the prediction. How good is the prediction?

\[
\begin{align*}
\text{In [27]: } & \quad \text{model} = \text{skl\_lm.LinearRegression()} \\
& \quad \text{model.fit(x.reshape(-1,1), y.reshape(-1,1))} \quad \quad \text{# reshape because the model requires input to be a 2D-array} \\
& \quad \text{prediction} = \text{model.predict(x.reshape(-1,1))} \\
& \quad \text{prediction_class} = \text{np.repeat}(0, 40) \\
& \quad \text{prediction_class[np.squeeze(prediction>=0.5)]} = 1
\end{align*}
\]
3.3 (c)

Try instead logistic regression using \texttt{skl\_lm.LogisticRegression()} command (set the parameter \texttt{C} to 1000) and plot the prediction. How good is the prediction, and what advantages does logistic regression have over linear regression for this classification problem?

In [28]: # c)
model = skl\_lm.LogisticRegression(C=1000)
model.fit(x.reshape(-1,1), y)
prediction = model.predict\_proba(x.reshape(-1,1))
prediction\_class = np.repeat(0, 40)
prediction\_class[prediction[:, 1]>=0.5] = 1
plt.plot(x, prediction[:, 1], label='logistic regression')
plt.scatter(x, y, facecolors='none', edgecolors='r', label='data')
plt.scatter(x, prediction\_class+0.02, facecolors='none', edgecolors='b',
        label='thresholded log. reg.')
plt.xlim(0, 40)
plt.legend()
plt.show()
4 4.4 k-NN

In this exercise, we are going to explore an important user aspect of $k$-NN.

4.1 (a)

Make 200 draws $x_1$ from a $\mathcal{N}(0, 1^2)$ distribution, and 200 draws $x_2$ from $\mathcal{N}(0, 10^4)$. Also construct $y$ such that $y = 1$ if $x_1 \cdot x_2$ is positive, and 0 otherwise. Split the data set randomly into a test and a training data set (equally sized).

In [29]: # a)
    np.random.seed(2)
    N = 200
    x1 = np.random.normal(0, 1, 200)
    x2 = np.random.normal(0, 100, 200)

    y = np.repeat(0, N)
    y[x1*x2 > 0] = 1

    X = pd.DataFrame({'x1': x1, 'x2': x2})

    X_train = X.iloc[:100, :]
    X_test = X.iloc[-100:, :]
    y_train = y[:100]
    y_test = y[-100:]
4.2 (b)

Use \( k \)-NN (choose \( k \) yourself) to predict the test output \( y \) using \( x_1 \) and \( x_2 \) as inputs. How well do you perform?

\[\text{In } [30]: \quad \text{# b)}\]
\[
\text{model} = \text{skl_nb.KNeighborsClassifier(n_neighbors=2)}
\text{model.fit(X_train, y_train)}
\text{prediction} = \text{model.predict(X_test)}
\text{np.mean(prediction == y_test)}
\]
\[\text{Out}[30]: \quad 0.52\]

4.3 (c)

Now replace \( x_2 \) with 200 draws from \( N(0,1^2) \), and perform \( k \)-NN classification anew. How well do you perform this time? Explain the difference!

\[\text{In } [31]: \quad \text{# c)}\]
\[
\text{np.random.seed(2)}
\text{N} = 200
\text{x1} = \text{np.random.normal(size=N)}
\text{x2} = \text{np.random.normal(size=N)}

\text{y} = \text{np.repeat(0, N)}
\quad \text{# y}[x1*x2 > 0] = 1
\text{y} = 1*(x1*x2>0)

\text{X} = \text{pd.DataFrame(\{'X1': x1, 'X2': x2\})}
\text{X_train} = \text{X.iloc[:100, :]}
\text{X_test} = \text{X.iloc[-100:, :]}
\text{y_train} = \text{y[:100]}
\text{y_test} = \text{y[-100:]}\]
\[
\text{model} = \text{skl_nb.KNeighborsClassifier(n_neighbors=3)}
\text{model.fit(X_train, y_train)}
\text{prediction} = \text{model.predict(X_test)}
\text{np.mean(prediction == y_test)}
\]
\[\text{Out}[31]: \quad 0.92\]

\[\text{In } [32]: \quad \text{# k-NN is based on the Euclidian distance between data points. In our}
\text{# problem in (b), the values of x2 is on average 100 times larger than}
\text{# the values of x1, and hence does the prediction essentially only}
\text{# depend on x2 (e.g., the distance between (0.1,10) and (0.1,-10)}
\text{# is larger than the distance between (0.1,10) and (-0.1,-9),}
\text{# e.g., x1 does effectively not matter when determining the k nearest}
\text{# neighbors). However, since y depends both # on x1 and x2, the} \]
# performance is deteriorated. Now, when removing the magnitude # difference between x1 and x2, both inputs will impact the k-NN # prediction equally.

4.4 (d)

Explore how the `sklearn.preprocessing.scale()` function can help for such problems encountered in (b)!

In [33]: np.random.seed(2)
   N = 200
   x1 = np.random.normal(0, 1, 200)
   x2 = np.random.normal(0, 100, 200)
   x2 = skl_pre.scale(x2)  # Normalize X2

   y = np.repeat(0, N)
   y[x1*x2 > 0] = 1

   X = pd.DataFrame({'x1': x1, 'x2': x2})

   X_train = X.iloc[:100, :]
   X_test = X.iloc[-100:, :]
   y_train = y[:100]
   y_test = y[-100:]

   model = skl_nb.KNeighborsClassifier(n_neighbors=2)
   model.fit(X_train, y_train)
   prediction = model.predict(X_test)

   np.mean(prediction == y_test)

Out[33]: 0.92

5 4.5 Multiclass classification

In the course, we have focused on the classification problem for 2 classes. The methods can, however, be generalized to more than two classes. In Python, the commands `skl_da.LinearDiscriminantAnalysis()`, `skl_da.QuadraticDiscriminantAnalysis()` and `skl_nb.KNeighborsClassifier()` can all be used directly for multi-class problems as well, which we will do in this exercise.

5.1 (a)

Load and familiarize yourself with the data set `iris`, and split it randomly into a training and a test data set.

Description
This famous (Fisher’s or Anderson’s) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

**Format**

iris is a data frame with 150 cases (rows) and 5 variables (columns) named Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.

In [34]: np.random.seed(1)
   ...: iris = pd.read_csv('Data/iris.csv')
   ...: iris.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
  #   Name    Non-Null Count  Dtype
  ---  ------   --------------  -----  
  0   Sepal.Length  150 non-null float64
  1   Sepal.Width    150 non-null float64
  2   Petal.Length   150 non-null float64
  3   Petal.Width    150 non-null float64
  4   Species        150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB

In [35]: # sampling indices for training
   ...: trainI = np.random.choice(iris.shape[0], size=100, replace=False)
   ...: trainIndex = iris.index.isin(trainI)
   ...: iris_train = iris.iloc[trainIndex] # training set
   ...: iris_test = iris.iloc[-trainIndex] # test set

5.2 (b)

Use all inputs (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width) to predict the output Species (setosa, versicolor and virginica) using LDA, QDA, and k-NN, respectively.

In [36]: input_variables = ['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']

   ...: X_train = iris_train[input_variables]
   ...: Y_train = iris_train['Species']
   ...: X_test = iris_test[input_variables]
   ...: Y_test = iris_test['Species']

5.2.1 LDA

In [37]: model = skl_da.LinearDiscriminantAnalysis()
   ...: model.fit(X_train, Y_train)
   ...: prediction = model.predict(X_test)
   ...: np.mean(prediction == Y_test)

Out[37]: 0.96
5.2.2 QDA

In [38]: model = skl_da.QuadraticDiscriminantAnalysis()
   model.fit(X_train, Y_train)

   prediction = model.predict(X_test)
   np.mean(prediction == Y_test)

Out[38]: 0.94

5.2.3 KNN

In [39]: model = skl_nb.KNeighborsClassifier(n_neighbors=2)
   model.fit(X_train, Y_train)

   prediction = model.predict(X_test)
   np.mean(prediction == Y_test)

Out[39]: 0.9