Introduction_to_Machine_Learning

October 28, 2023

#CSE422 Lab ## Introduction to Machine Learning: From Theory to Application —

Background Information: Iris is a family of multiple species of flowering plants. Setosa is one species of flower, Versicolor is another, and virginica is another.

Problem statement: Suppose you are a botany student and while doing botany classes all year round you came to know that some distinguishing features of flower species are **sepal length and width, and petal length and width**. You also came across a set of samples in your class extracted from the Amazon rainforest that have already been classified into their appropriate species by experts.

In your final examination, you are taken to the Amazon rainforest and given the task of predicting the species of a set of new flowers that have not yet been classified.

0.1 ###The Statistical Learning Framework

1) The learner's input: A learner/algorithm has access to the following:

- **Domain set**: An arbitrary set, X where $x_i \in \mathbb{R}^d$ and d is the number of features. This is the set of objects that we may wish to label. For example, for the iris dataset, the domain set will be the set of all flowers. Usually, these domain points will be represented by a vector of features like a flower's petal width, petal length, and so on. We also refer to domain points as instances/samples and to X as instance/sample space.
- **Label set**: For the iris dataset, the label set is a three-element set: $\{0, 1, 2\}$. Let Y denote our set of possible labels: {0, 1, 2}, where 0 represents the flower type 'setosa', 1 represents the flower type 'versicolor', and 2 represents 'virginica'.
- **Training data**: $S = \{(x_1, y_1), (x_2, y_2)....(x_m, y_m)\}$ is a finite sequence of pairs in $X \times Y$: that is, a sequence of labeled domain points. This is the input that the learner has access to. In our case it is the set of flowers for which petal length, petal width, sepal length and sepal width have been measured and their flower type determined by botanists. Such labeled examples are often called training examples. We sometimes also refer to S as a training set.

2) The learner's output: The learner outputs a prediction rule, $h : X \to Y$. One can think of the rule as a function. This function is also called a predictor, or a hypothesis. The predictor can be used to predict the label of new domain points. In our iris dataset, it is a rule that our learner will employ to predict whether future flowers it examines are going to be setosa, virginica or versicolor. Upon receiving a training sequence S a learning algorithm returns a hypothesis.

A simple data-generation model Let us now understand how the training data is generated. First, we assume that the instances (the flowers we encounter) are generated by some probability distribution (in this case, representing the environment). Let D be the probability distribution over X. It is important to note that we do not assume that the learner knows anything about this distribution. As to the labels, in the current discussion we assume that there is some "correct" labeling function, $f : X \rightarrow Y$, and that $y_i = f(x_i)$ for all i. The labeling function is unknown to the learner. In fact, this is just what the learner is trying to figure out. In summary, each pair in the training data S is generated by first sampling a point x_i according to D and then labeling it by f.

3) Measures of success: We define the error of a classifier to be the probability that it does not predict the correct label on a random data point generated by the aforementioned underlying distribution, D. That is, the error of h is the probability of drawing a random instance x, according to the distribution D, such that $h(x)$ does not equal $f(x)$.

We define the error of a prediction rule, \$ h : X → Y \$, to be:

$$L_{D,f}(h) = P_{x \sim D}[h(x) \neq f(x)]$$

That is, the error of such h is the probability of randomly choosing an example x for which $h(x) \neq f(x)$. The subscript (D, f) indicates that the error is measured with respect to the probability distribution D and the correct labeling function f. $L_{(D,f)}(h)$ has several synonymous names such as the generalization error, the risk, the true error of h, or the loss of the learner.

Note on the information available to the learner: The learner is blind to the underlying distribution D over the Amazon rainforest and to the labeling function f. In our iris example, we have just arrived in the Amazon rainforest and we have no clue as to how the flowers are distributed and how to predict the species they fall under. The only way the learner can interact with the environment is through observing the training set(which was seen in the student's classroom).

This form of learning is called Supervised Learning since the learner is provided with labels (y_i, where $i = 0, 1, 2, ..., m$) while training.

Reference: Shalev-Shwartz, S., & Ben-David, S. (2017). Understanding Machine learning: from theory to algorithms. Cambridge: Cambridge University Press.

0.2 A First Application: Classifying Iris Species

Meet the Data

[ ]:
```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

[ ]:
```
print(type(itris_datase))
```

```
<class 'sklearn.utils.Bunch'>
```

[ ]:
```
print("Keys of iris_dataset:\n", iris_dataset.keys())
```

```
Keys of iris_dataset:
 dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

[ ]:
```
iris_dataset
```

[ ]:
```
{'DESCR': '.. _iris_dataset:\n\nIris plants
dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of
Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4
numeric, predictive attributes and the class\n    :Attribute Information:\n
- sepal length in cm\n        - sepal width in cm\n        - petal length in
cm\n        - petal width in cm\n        - class:\n                - Iris-
Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n
\n    :Summary Statistics:\n\n    ============== ==== ==== ======= =====
====================\n                    Min  Max   Mean    SD   Class
Correlation\n    ============== ==== ==== ======= ===== ====================\n
sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4
3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490
(high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n
============== ==== ==== ======= ===== ====================\n\n    :Missing
Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n
:Creator: R.A. Fisher\n    :Donor: Michael Marshall
(MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris
database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s
paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning
Repository, which has two wrong data points.\n\nThis is perhaps the best known
database to be found in the\npattern recognition literature.  Fisher\'s paper is
a classic in the field and\nis referenced frequently to this day.  (See Duda &
Hart, for example.)  The\ndata set contains 3 classes of 50 instances each,
where each class refers to a\ntype of iris plant.  One class is linearly
separable from the other 2; the\nlatter are NOT linearly separable from each
other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple
measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188
(1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley,
NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and
Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See
page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New
System\n     Structure and Classification Rule for Recognition in Partially
Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and
Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972)
"The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information
Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.
Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3
classes in the data.\n   - Many, many more …',
 'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
        [4.8, 3. , 1.4, 0.3],
        [5.1, 3.8, 1.6, 0.2],
        [4.6, 3.2, 1.4, 0.2],
        [5.3, 3.7, 1.5, 0.2],
        [5. , 3.3, 1.4, 0.2],
        [7. , 3.2, 4.7, 1.4],
        [6.4, 3.2, 4.5, 1.5],
        [6.9, 3.1, 4.9, 1.5],
        [5.5, 2.3, 4. , 1.3],
        [6.5, 2.8, 4.6, 1.5],
        [5.7, 2.8, 4.5, 1.3],
        [6.3, 3.3, 4.7, 1.6],
        [4.9, 2.4, 3.3, 1. ],
        [6.6, 2.9, 4.6, 1.3],
        [5.2, 2.7, 3.9, 1.4],
        [5. , 2. , 3.5, 1. ],
        [5.9, 3. , 4.2, 1.5],
        [6. , 2.2, 4. , 1. ],
        [6.1, 2.9, 4.7, 1.4],
        [5.6, 2.9, 3.6, 1.3],
        [6.7, 3.1, 4.4, 1.4],
        [5.6, 3. , 4.5, 1.5],
        [5.8, 2.7, 4.1, 1. ],
        [6.2, 2.2, 4.5, 1.5],
        [5.6, 2.5, 3.9, 1.1],
        [5.9, 3.2, 4.8, 1.8],
        [6.1, 2.8, 4. , 1.3],
        [6.3, 2.5, 4.9, 1.5],
        [6.1, 2.8, 4.7, 1.2],
        [6.4, 2.9, 4.3, 1.3],
        [6.6, 3. , 4.4, 1.4],
        [6.8, 2.8, 4.8, 1.4],
        [6.7, 3. , 5. , 1.7],
        [6. , 2.9, 4.5, 1.5],
        [5.7, 2.6, 3.5, 1. ],
        [5.5, 2.4, 3.8, 1.1],
        [5.5, 2.4, 3.7, 1. ],
        [5.8, 2.7, 3.9, 1.2],
        [6. , 2.7, 5.1, 1.6],
        [5.4, 3. , 4.5, 1.5],
        [6. , 3.4, 4.5, 1.6],
        [6.7, 3.1, 4.7, 1.5],
        [6.3, 2.3, 4.4, 1.3],
        [5.6, 3. , 4.1, 1.3],
        [5.5, 2.5, 4. , 1.3],
        [5.5, 2.6, 4.4, 1.2],
        [6.1, 3. , 4.6, 1.4],
        [5.8, 2.6, 4. , 1.2],
        [5. , 2.3, 3.3, 1. ],
        [5.6, 2.7, 4.2, 1.3],
        [5.7, 3. , 4.2, 1.2],
        [5.7, 2.9, 4.2, 1.3],
        [6.2, 2.9, 4.3, 1.3],
        [5.1, 2.5, 3. , 1.1],
        [5.7, 2.8, 4.1, 1.3],
        [6.3, 3.3, 6. , 2.5],
        [5.8, 2.7, 5.1, 1.9],
        [7.1, 3. , 5.9, 2.1],
        [6.3, 2.9, 5.6, 1.8],
        [6.5, 3. , 5.8, 2.2],
        [7.6, 3. , 6.6, 2.1],
        [4.9, 2.5, 4.5, 1.7],
        [7.3, 2.9, 6.3, 1.8],
        [6.7, 2.5, 5.8, 1.8],
        [7.2, 3.6, 6.1, 2.5],
        [6.5, 3.2, 5.1, 2. ],
        [6.4, 2.7, 5.3, 1.9],
        [6.8, 3. , 5.5, 2.1],
        [5.7, 2.5, 5. , 2. ],
        [5.8, 2.8, 5.1, 2.4],
        [6.4, 3.2, 5.3, 2.3],
        [6.5, 3. , 5.5, 1.8],
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'data_module': 'sklearn.datasets.data',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'frame': None,
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10')}
```

Knowing Your Task and Knowing Your Data

```
[ ]: #print(iris_dataset['DESCR'])
```

```
[ ]: print("Feature names:\n", iris_dataset['feature_names'])
```

```
Feature names:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```

```
[ ]: print("Target names:", iris_dataset['target_names'])
```

```
Target names: ['setosa' 'versicolor' 'virginica']
```

```
[ ]: print("Type of data:", type(iris_dataset['data']))
```

```
Type of data: <class 'numpy.ndarray'>
```

```
[ ]: print("Shape of data:", iris_dataset['data'].shape)
```

```
Shape of data: (150, 4)
```

```
[ ]: print("First five rows of data:\n", iris_dataset['data'][:10])
```

```
First five rows of data:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

[ ]:
```python
print("Type of target:", type(iris_dataset['target']))
```

```
Type of target: <class 'numpy.ndarray'>
```

[ ]:
```python
print("Shape of target:", iris_dataset['target'].shape)
```

```
Shape of target: (150,)
```

We can see that there are 3 classes of target labels. i.e. the flowers have been categorized into 3 classes.

[ ]:
```python
print("Target:\n", iris_dataset['target'])
```

```
Target:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

0.2.1 What is Scikit learn?

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support k-nearest neighbours, support vector machines, random forests, gradient boosting and so on, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. It also has other modules with helpful functions used in machine learning projects such as `train_test_split`.

```
                                                    --from Wikipedia
```

Measuring Success: Training and Testing Data

- Why do we train_test_split?
- Stratified v.s. Random Split

[ ]:
```python
from sklearn.model_selection import train_test_split
#random data splitting
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],␣
 ↪iris_dataset['target'], test_size = 0.25, random_state=0)
```

[ ]:
```python
#Stratified data splitting
#y = pd.DataFrame(iris_dataset['target'])
```

```
#X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],␣
↪iris_dataset['target'], test_size = 0.25, random_state=0, stratify = y)
```

[ ]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (112, 4)
y_train shape: (112,)
```

[ ]:
```
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

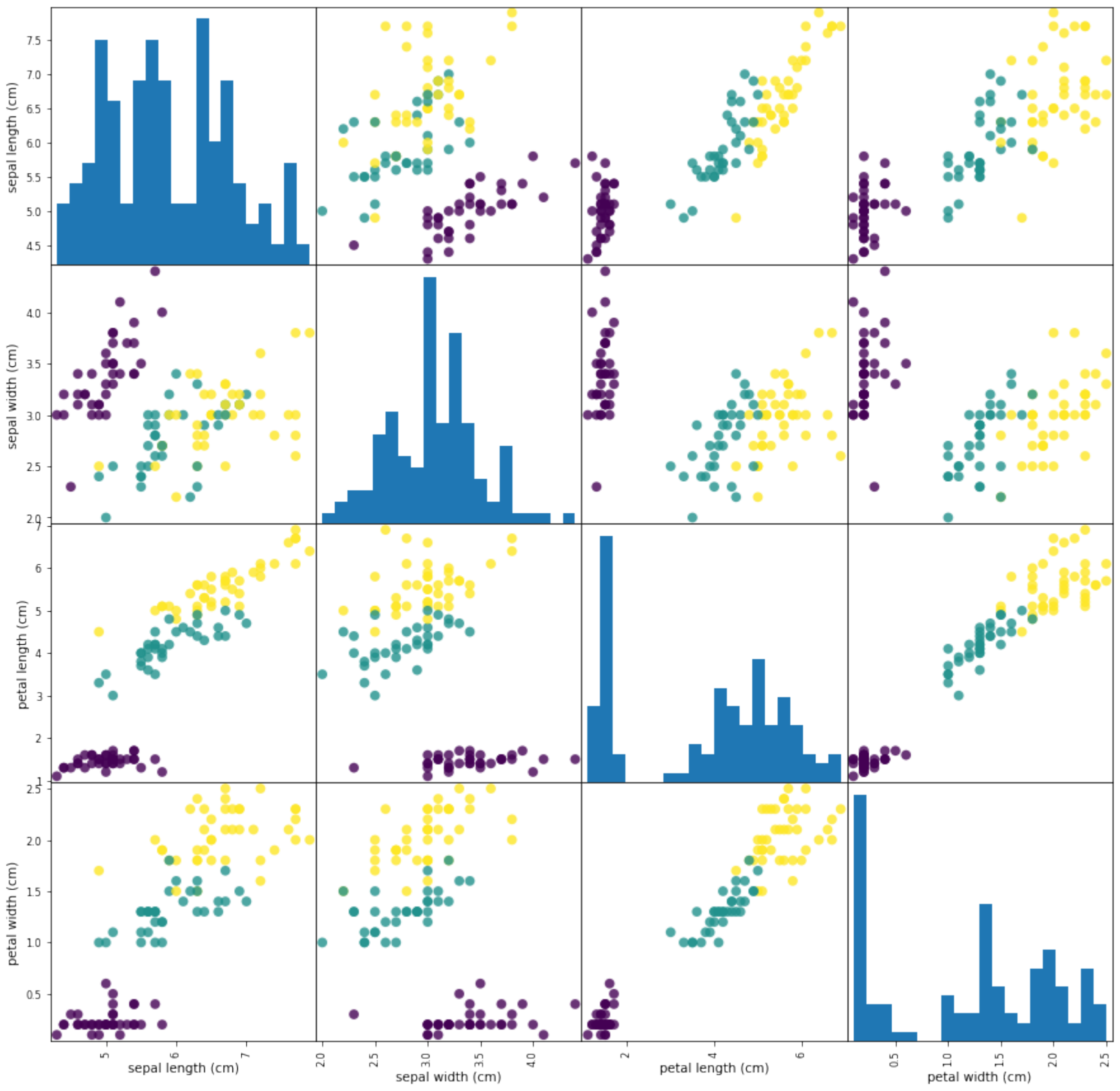
```
X_test shape: (38, 4)
y_test shape: (38,)
```

Visualizing your data to gain more insight.

From the below scatter matrix, we can see that one of the class is linearly separable from the other 2, but the other 2 are not linearly separable from each other. So it's better if we use a non-linear model like KNN. ####It is important to understand how your data is distributed and use a suitable algorithm accordingly.

[ ]:
```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
                           marker='o', hist_kwds={'bins': 20}, s=60,
                           alpha=.8)
```

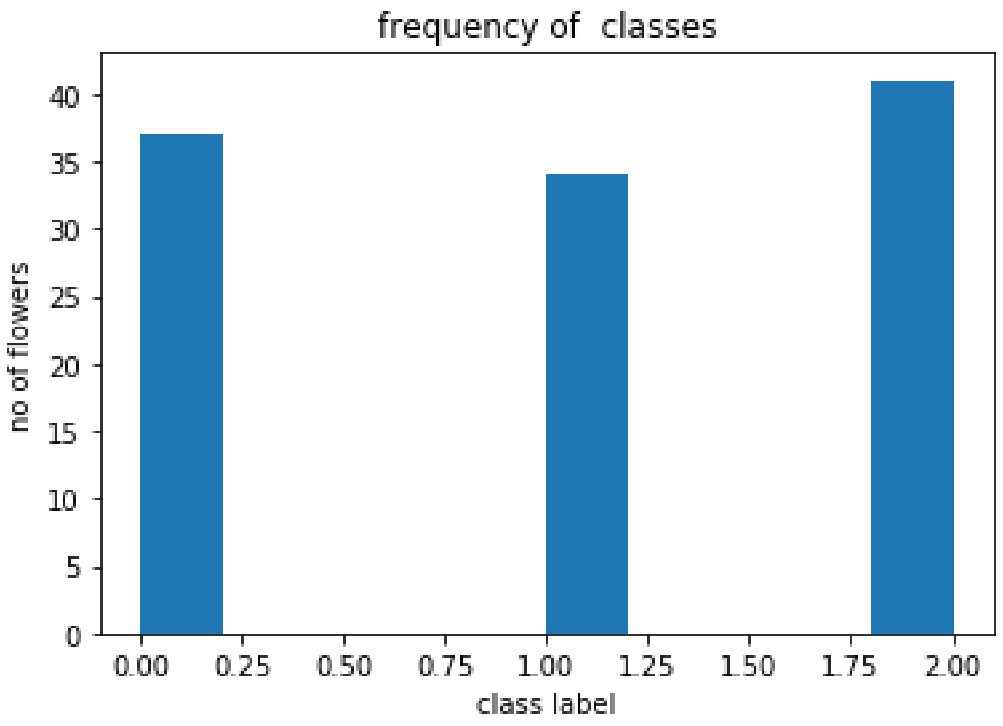
[ ]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb9b8ca58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb98b7be0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb986dd68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb98a1f28>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb985f4e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb980da90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb97cb080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb977a668>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb977a6a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb96e91d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb9718780>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb96cad30>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb9687320>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb96368d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb95e7e80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0eb9625470>]],
      dtype=object)
```



- Now, we will check if there are somewhat equal number of samples for each class. Otherwise, if there is a big shortage of samples of one class, my model might not learn to identify that specific class.
- 0 - iris-setosa, 1 - iris-versicolor, 2 - iris-virginica

```
[ ]: fig,ax = plt.subplots()
     ax.hist(y_train)
     ax.set_title('frequency of  classes')
     ax.set_xlabel('class label')
     ax.set_ylabel('no of flowers')
```

```
[ ]: Text(0, 0.5, 'no of flowers')
```



Building and Evaluating Your First Model: k-Nearest Neighbors

- KNN is a non-linear classifier; exactly what we need!

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
```

[ ]:
```
knn.fit(X_train, y_train)
```

[ ]:
```
KNeighborsClassifier(n_neighbors=3)
```

0.2.2 Making Predictions

####Just giving a sample of 1 flower(by passing an array of feature values) to check what my model categorizes it as.

[ ]:
```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape:", X_new.shape)
```

```
X_new.shape: (1, 4)
```

[ ]:
```
prediction = knn.predict(X_new)
print("Prediction:", prediction)
print("Predicted target name:",
       iris_dataset['target_names'][prediction])
```

```
Prediction: [0]
Predicted target name: ['setosa']
```

Evaluating my Model on the test data: X_test

```
[ ]: y_pred = knn.predict(X_test)
     print("Test set predictions:\n", y_pred)
```

```
Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```

Manually checking my error and accuracy—just to understand what is happening

```
[ ]: count = 0
     for i in range( len(y_test) ):
         if y_pred[i] != y_test[i]:
             count = count + 1
     error = count/len(y_pred)
     print( "Error = %f " % (error*100) + '%' )
     accuracy = (1-error)
     print( "Accuracy = %f " % (accuracy*100) + '%' )
```

```
Error = 2.631579 %
Accuracy = 97.368421 %
```

Some easier ways to calculate accuracy.

```
[ ]: print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.97
```

The `knn.score()` function takes in X_test, calls the `predict` function and then calls the `accuracy_score` function.

Basically, carries the two steps of: 1) calculating prediction and 2) evaluation the model, with one function call.

```
[ ]: print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

0.2.3 Outlook

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(
         iris_dataset['data'], iris_dataset['target'], random_state=0)

     knn = KNeighborsClassifier(n_neighbors=1)

     knn.fit(X_train, y_train)

     print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```

0.2.4 Summary

1) Split your data set into train and test sets.
2) Create the model object.
3) Train(using `.fit(X_train, y_train)`) your ML model on training set.
4) Calculate the predictions (using `.predict(X_test)`) of data in the test set.
5) Evaluate the predictions by calculating the accuracy on the test set.

.

.

Reference: Müller Andreas Christian, and Sarah Guido. Introduction to Machine Learning with Python a Guide for Data Scientists. OReilly, 2018.