Machine Learning

Lesson 12

Implementation of Machine Learning Algorithms in Python

Lecturer: Dr. Msagha J Mbogholi, PhD

Flashback from Lesson 11

- Within the same domain you may have to use a different algorithm than the one favored in the literature; this brings about the need to have a way by which an evaluation of a suitable algorithm among many can be done.
- The algorithm and classifier problem can be broken down into 3 sub domains namely classification, learning and evaluation
- Levasson (2006) proposed a framework by which evaluation can be done; the 4 methods proposed are : algorithm dependent, algorithm independent, classifier independent, and classifier dependent.
- Other evaluation tests include f test, accuracy, precision, recall, error rate, Nemenyi test and Friedman test.
- In Python sklearn provides a platform to test and evaluate algorithms; this can be done by determining a scoring strategy and implementing it in the kit.

Content

- Python data analysis packages
- Hello world Machine learning project

Part 1

Python data analysis packages

Introduction

- In lesson 11 we had a sneak preview of sckit, arguably the most popular library in use by machine learning community.
- It is not the only one as shall be seen in this lesson.
- This is the final lesson in this course; it is certainly not everything there is to know about machine learning (we need about two more courses to go into enough detail and explanation for that) but it is enough to have satisfied your curiosity about ML and hopefully, get you excited enough to dig deeper on your own.
- In this lesson we examine the data analysis packages in Python and discuss your first ML project as demonstrated by Brownlee (2020). The three main data analysis packages are Numpy, Pandas, and Matplotlib.
- Fig 1 shows the genesis of their growth and their specialties.

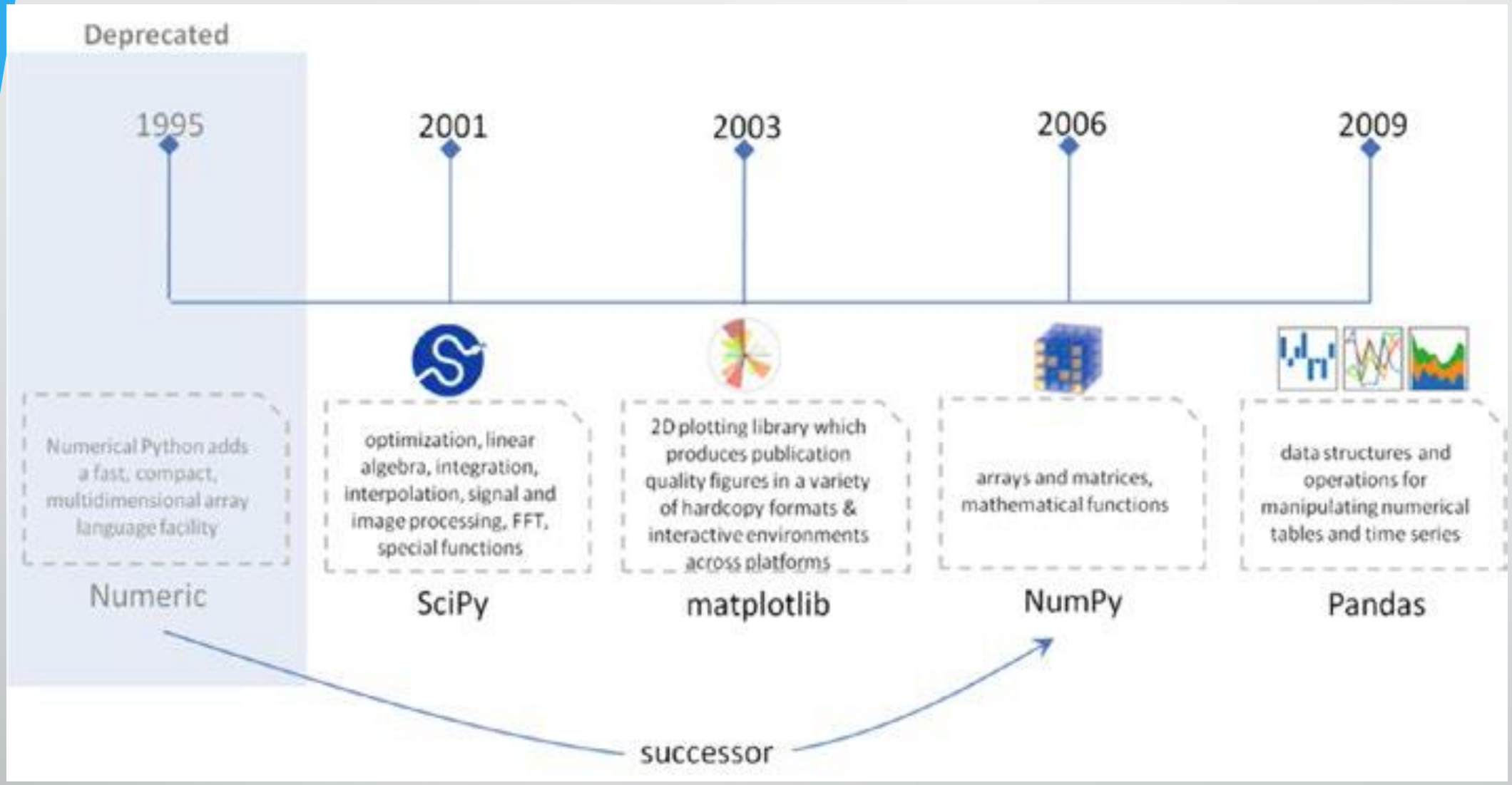

Fig 1 Genesis and description of data analysis packages (Swamynathan, 2017)

1.1 Numpy

- This is the core library for scientific computing (Swamynathan, 2017).
- It provides functionality to work with arrays from two dimensions to multidimensional arrays.
- Some of the operations it performs on arrays include: creating, slicing, indexing and broadcasting.
- I assume that you have at least elementary knowledge of Python or some other programming language?
- If not don't worry; Python is fairly intuitive and should not be hard to follow; most of the commands are written almost in English.
- Let us demonstrate a few operations using numpy that are useful in machine learning.

1.1.1 Some numpy operations

- Create a numpy array of all 2's:
- # Create a 3x3 array of all twos
- a = np.twos((3,3))
- print a
- ----- output -----
- [[2. 2. 2.]
- [2. 2. 2.]
- [2. 2. 2.]]

- To use random numbers:
- # Create a 3x3 array filled with random values
- d = np.random.random((3,3))
- print d
- ---- output ----
- [[0.85536712 0.14369497 0.46311367]
- [0.78952054 0.43537586 0.48996107]
- [0.1963929 0.
- 12326955 0.00923631]]

- Choosing data types:
- # Let numpy choose the datatype
- x = np.array([0, 3])
- y = np.array([2.6, 3.8])
- # Force a particular datatype
- z = np.array([9, 10], dtype=np.int64)
- print x.dtype, y.dtype, z.dtype
- ---- output ----
- int32 float64 int64

1.1.1 Some numpy operations

- Slicing arrays is same as slicing a list in Python:
- x = np.array([5, 6, 7, 8, 9])
- print x[-2:5]
- print x[-1:1:-1]
- # ---- output ----
- [8 9]
- [9 8 7]
- # Create a rank 2 array with shape (3, 4)
- a = np.array([[5,6,7,8], [1,2,3,4], [9,10,11,12]])
- print "Array a:", a
- # Use slicing to pull out the subarray consisting of the first 2 rows
- # and columns 1 and 2; b is the following array of shape (2, 2):
- # [[2 3]
- # [6 7]]
- b = a[:2, 1:3]
- print "Array b:", b
- ---- output ----
- Array a: [[5 6 7 8]
- [1 2 3 4]
- [9 10 11 12]]
- Array b: [[6 7]
- [2 3]]
- A slice of an array

1.1.1 Some numpy operations

- Mathematical operations are performed element wise in numpy arrays:
- x=np.array([[1,2],[3,4],[5,6]])
- y=np.array([[7,8],[9,10],[11,12])
- # Elementwise sum; both produce the array
- printx+y
- printnp.add(x, y)
- # ---- output ----
- [[8 10]
- [12 14]
- [16 18]]
- [[8 10]
- [12 14]
- [16 18]]
- # Elementwise difference; both produce the array
- printx-y
- printnp.subtract(x, y)
- # ---- output ----
- [[-4. -4.]
- [-4. -4.]]
- [[-4. -4.]
- [-4. -4.]]

1.1.1 Some numpy operations

- # Elementwise product; both produce the array
- printx*y
- printnp.multiply(x, y)
- # ---- output ----
- [[5. 12.]
- [21. 32.]]
- [[5. 12.]
- [21. 32.]]
- # Elementwise division; both produce the array
- printx/y
- printnp.divide(x, y)
- # ---- output ----
- [[0.2 0.33333333]
- [0.42857143 0.5]]
- [[0.2 0.33333333]
- [0.42857143 0.5]]
- # Elementwise square root; produces the array
- printnp.sqrt(x)
- # ---- output ----
- [[1. 1.41421356]
- [1.73205081 2.]]
- These examples are from Swamynathan (2017)

1.2 Pandas

- As described in fig 1 Pandas works with data structures and is used for manipulations of tables and time series.
- The data structures are series and DataFrame.
- Additionally other operations that can be performed include: viewing data, merging/joining, presenting summaries and statistics, reading and writing data.
- Due to time and space we shall only look at some of the functionalities to give you an idea of how they work.

1.2.1 Panda operations

- # creating a series by passing a list of values, and a custom index label.
- Note that the labeled index reference for each row and it can have duplicate
- values
- s = pd.Series([1,2,3,np.nan,5,6], index=['A','B','C','D','E','F'])
- print s
- # ---- output ----
- # A 1.0
- # B 2.0
- # C 3.0
- # D NaN
- # E 5.0
- # F 6.0
- # dtype: float64

1.2.1 Pandas operations

- #how to create a pandas dataframe
- data = {'Gender': ['F', 'M', 'M'],'Emp_ID': ['E01', 'E02', 'E03'], 'Age': [25, 27, 25]}
- # We want the order the columns, so lets specify in columns parameter
- df = pd.DataFrame(data, columns=['Emp_ID','Gender', 'Age'])
- df
- # ---- output ----
- # Emp_ID Gender Age
- #0 E01 F 25
- #1 E02 M 27
- #2 E03 M 25

- # reading from multiple sheets of same Excel into different dataframes
- xlsx = pd.ExcelFile('file_name.xls')
- sheet1_df = pd.read_excel(xlsx, 'Sheet1')
- sheet2_df = pd.read_excel(xlsx, 'Sheet2')
- # writing
- # index = False parameter will not write the index values, default is True
- df.to_csv('Data/mtcars_new.csv', index=False)
- df.to_csv('Data/mtcars_new.txt', sep='\t', index=False)
- df.to_excel('Data/mtcars_new.xlsx',sheet_name='Sheet1', index = False)

1.2.1 Pandas operations

- Let us examine a few statistics that pandas (pd) provides: this is from the iris datasheet which we shall revisit in our "hello word" machine program. The statistics here are provided from Swamynathan (2017). This makes use of the dataframe (df):
- df = pd.read_csv('Data/iris.csv')
- df.describe()
- #---- output ----
- #Sepal.Length Sepal.Width Petal.Length Petal.Width
- #count 150.000000 150.000000 150.000000 150.000000
- #mean 5.843333 3.057333 3.758000 1.199333
- #std 0.828066 0.435866 1.765298 0.762238
- #min 4.300000 2.000000 1.000000 0.100000
- #25% 5.100000 2.800000 1.600000 0.300000
- #50% 5.800000 3.000000 4.350000 1.300000
- #75% 6.400000 3.300000 5.100000 1.800000
- #max 7.900000 4.400000 6.900000 2.500000
- Finally let us calculate how our variables are related using covariance (cov). A positive covariance indicates that they are directly related, meaning as one in the pair increases so does the other one. A negative cov means they are inversely related.

1.2.1 Pandas operations

- df = pd.read_csv('Data/iris.csv')
- df.cov()
- #---- output ----
- #Sepal.Length Sepal.Width Petal.Length Petal.Width
- #Sepal.Length 0.685694 -0.042434 1.274315 0.516271
- #Sepal.Width -0.042434 0.189979 -0.329656 -0.121639
- #Petal.Length 1.274315 -0.329656 3.116278 1.295609
- #Petal.Width 0.516271 -0.121639 1.295609 0.581006

- Correlation is more specific than covariance, though both tell us how well a given pair are related. With correlation we work with an actual number in percentage form; for example 0.25 means 25% relation between the pair. Put simply this means that as one of the pair increases the other will increase by 25%. Let us apply this to our dataset:

1.2.1 Pandas operations

- df = pd.read_csv('Data/iris.csv')
- df.corr()
- #----output----
- # Sepal.Length Sepal.Width Petal.Length Petal.Width
- #Sepal.Length 1.000000 -0.117570 0.871754 0.817941
- #Sepal.Width -0.117570 1.000000 -0.428440 -0.366126
- #Petal.Length 0.871754 -0.428440 1.000000 0.962865
- #Petal.Width 0.817941 -0.366126 0.962865 1.000000

1.3 Matplotlib

- This is the mathematics package used to produce all those nice diagrams that you see in so many publications.
- It is generally used to plot lines, different types of charts in order for us to understand the data better; it also helps to quickly identify any trends that weren't apparent just by looking at the data.
- Let us examine a few examples to show how to use this package.

1.3.1 Matplotlib operations

- The following commands are used to plot the common types of charts and graphs:
- plt.bar – creates a bar chart
- plt.scatter – makes a scatter plot
- plt.boxplot – makes a box and whisker plot
- plt.hist – makes a histogram
- plt.plot – creates a line plot
- And now for some code that implements them.

1.3.1 Matplotlib operations

- # simple bar and scatter plot
- x = np.arange(5) # assume there are 5 students
- y = (20, 35, 30, 35, 27) # their test scores
- plt.bar(x,y) # Bar plot
- # need to close the figure using show() or close(), if not closed any follow
- #up plot commands will use same figure.
- plt.show() # Show the bar graph
- plt.scatter(x,y) # scatter plot
- plt.show() # show the scatter plot
- # ---- output ----
- # output is shown in fig 2 (a) and (b)

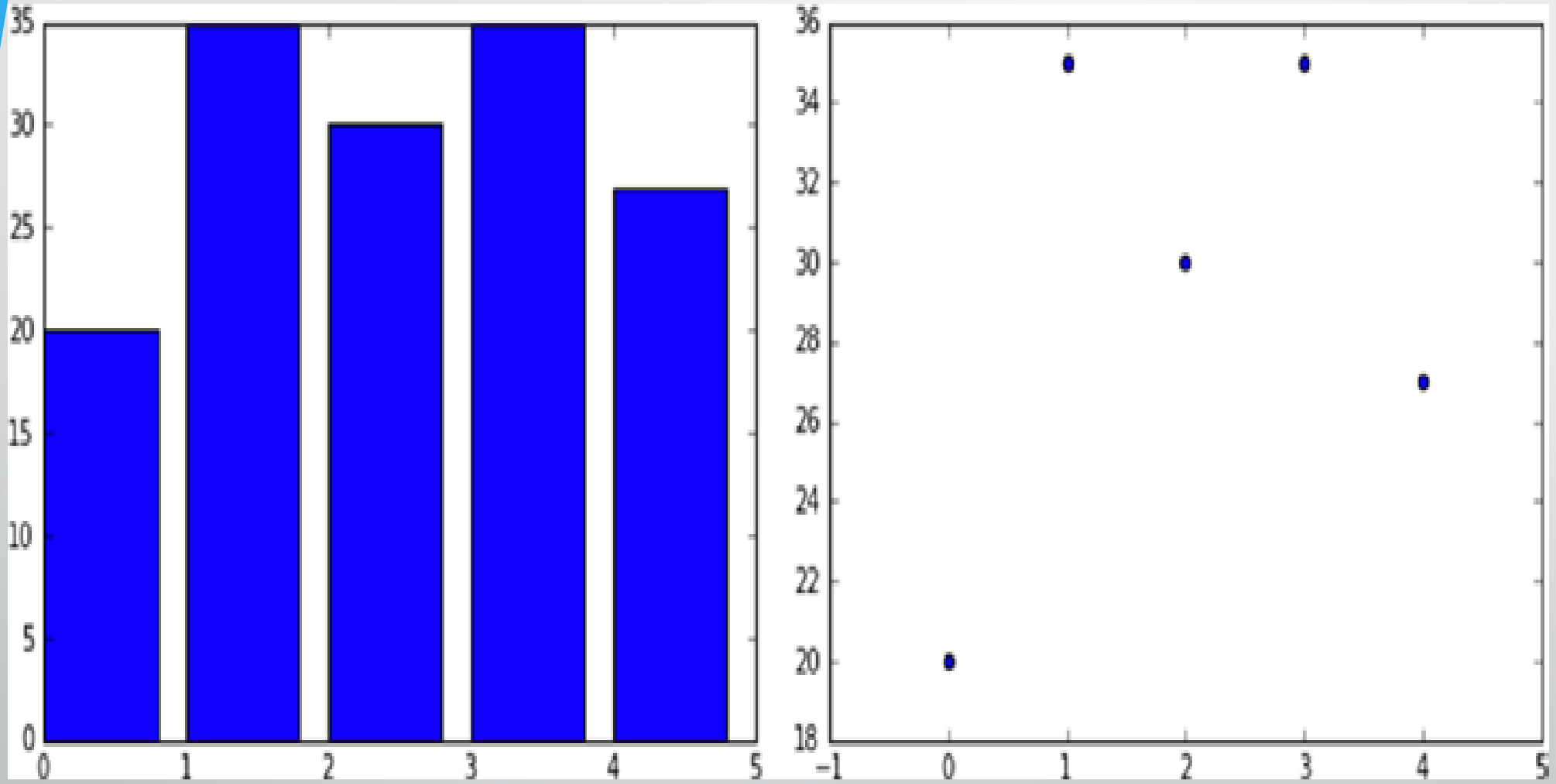

Fig 2 Bar chart (a) and scatter plot (b) (Swamynathan, 2017)

1.3.1 Matplotlib

- #let us extend this further using the same data to show a histogram, plot chart (line graph) and box chart, using the dataframe (df) object introduced earlier:
- df.hist()# Histogram
- df.plot() # Line Graph
- df.boxplot() # Box plot
- # the output is shown in fig 4 (a) for histogram, (b) for line graph and (c) for box plot

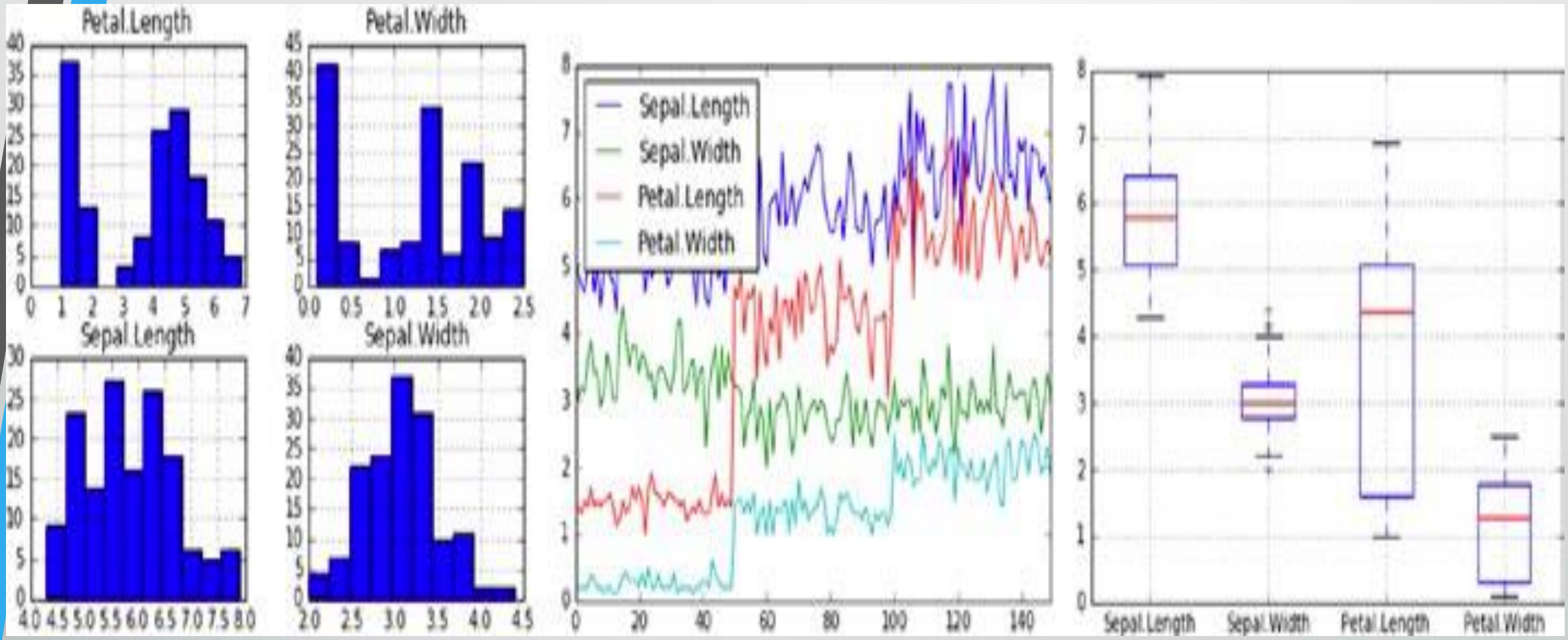

Fig 4 Histogram (a) and line graph (b) and box graph (c) (Swamynathan, 2017)

Part 2

Hello world machine learning project

Introduction

- If you have written a program in whichever language you are aware of what 'hello world' program is.
- In the first part of this lesson we introduced the main packages used for data analysis in machine learning using Python.
- This opens the door to you working through your first machine learning project.
- If you are like me you probably just went through part 1 breathlessly in order to get to this practical part.
- There are many online examples that guide one through the basics of creating a machine learning project.
- Of all the ones I have examined the one I have chosen to guide you through has been written by Brownlee (2020).
- We shall go through the steps of creating your first machine learning project so that you can build confidence to go further and do one of your own. This example is a very simple basic one.
- The dataset used for analysis is the common iris dataset you will have seen in many machine learning examples in literature and on the web. It is freely available at https://archive.ics.uci.edu/ml/datasets/Iris

Project steps

- Every machine learning project follows a number of steps from inception to conclusion. These are:
- Define the problem – you need to do a proper analysis of the problem at hand and understand it, much like you do in the software development life cycle (SDLC)
- Prepare the data – you will have some data to work with (otherwise would it really be a ML problem?); organize this data in a way that can be used in the ML environment.
- Evaluate algorithms – pick the best algorithm from the ones you had put under consideration; this could be by running the data through all of them in order to pick the one you deem most accurate in giving desired results
- Improve results – can the results be made more accurate by say combining algorithms or some other means? If so improve the results
- Present results - you would like the users to make sense of the data. Don't bamboozle them with results that won't make sense to them. Present the results clearly and in a manner that can be easily interpreted.

The Iris dataset

- This dataset is a popular dataset due to its many positive features, especially for learning how to use ML algorithms. In this course it is particularly helpful to use since it brings together all the concepts you have learnt in the past 11 lessons:
- Features are numeric making them easy to handle and load; it has 150 observations of 4 different features of iris flowers.
- It is a supervised learning problem involving classification; again it gives a soft landing since the algorithm is much simpler.
- It involves a multinomial distribution, implying that it is a multiclass classification
- It is a relatively small dataset
- All units are uniform requiring no complex mathematical transformations

Overview of the project

- There will be six steps to be covered:
    1. Install Python and sciPy
    2. Load dataset
    3. Examine dataset
    4. Visualize dataset
    5. Evaluate algorithms
    6. Make predictions
- We shall skip the first step: if you do not have python on your machine please follow the download instructions from www.python.org while the latter will be found at scipy.org . The latter contains the libraries discussed in part 1 of this lesson, plus sklearn which will also be needed.

2. Load dataset

- The file will be loaded from a csv URL
- In order to do so one must import all the necessary libraries; this is shown in fig 5.
- Then the dataset is loaded from its source; this is captured in fig 6.

```
# Load libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
...
```

Fig 5 Importing the libraries (Brownlee, 2020)

2. Loading the dataset

```
...
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
```

Fig 5 Loading the dataset (Brownlee, 2020)

3. Examining dataset

- In this next step we seek to analyse and understand the dataset better by:
  1. Examining the dimensions (attributes)
  2. The descriptive numeric data
  3. Getting a statistical summary of the data (introduced in part 1 of the lesson)
  4. Breaking the data down by class variable

3.1 Examining dimensions

- We get to see how many rows and columns exist in the data by using the shape command:
- print(dataset.shape), this will output: <150, 5> indicating that there are 150 rows and 5 columns.
- To see the data we use the head command, for example:
- # head
- print(dataset.head(20))
- will print the first 20 rows of the data; the output is depicted in fig 6

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

Fig 6 Output using the head command (Brownlee, 2020)

3.3 Statistical summary

- A statistical summary of the data is done using the describe command as follows:
- …
- # descriptions
- print(dataset.describe())
- The output is captured in fig 7.
- From the output we can see the mean, standard deviation, minimums and maximums, among other statistics of interest.

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Fig 7. Output description of data (Brownlee, 2020)

3.4 Breaking the data down by class variable

- This entails us seeing how the classes are distributed; that is, how many instances of each class exist in the dataset.
- We use the groupby command to group the data by class and size as follows:
- …
- # class distribution
- print(dataset.groupby('class').size())

- The output is as follows:
- class
- Iris-setosa 50
- Iris-versicolor 50
- Iris-virginica 50

4. Visualize dataset

- We perform visualization to understand the data better. In our hello world project we wish to do so from 2 perspectives:
- Plot each attribute to understand it better (univariate analysis)
- Understand the relationship between attributes (multivariate analysis)
- For the former we make use of a box and whisker plot to better understand each feature. The code to use is:
- …
- # box and whisker plots

  dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)

  pyplot.show()
- The output is shown in fig 8

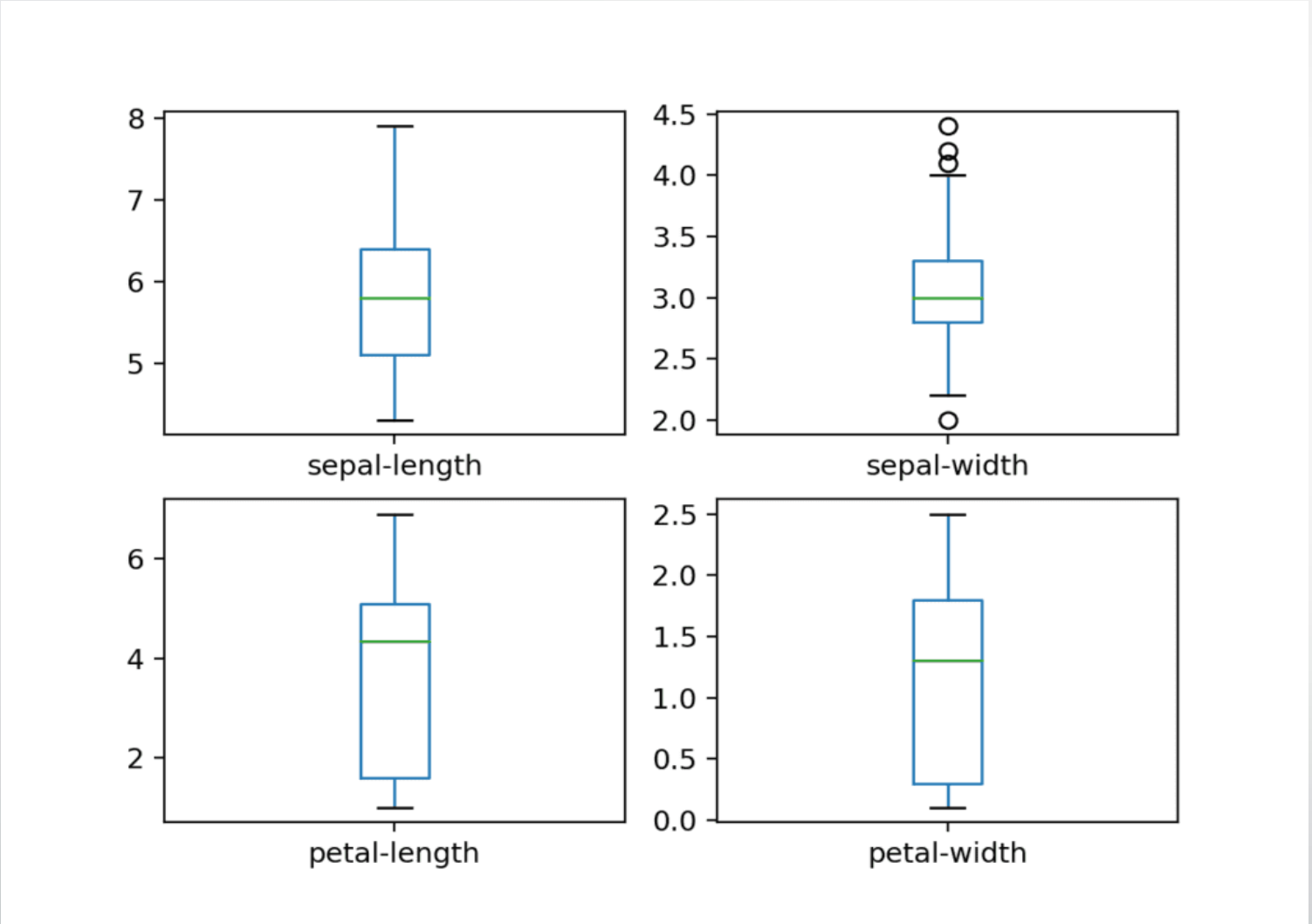

Fig 8 Box and whisker plot for each attribute (Brownlee, 2020)

4. Visualize dataset (cont'd)

- For the multivariate analysis we make use of a scatter plot matrix to see the relationship (if any) between the attributes.
- The command to do this is:
- …
- # scatter plot matrix

  scatter_matrix(dataset)

  pyplot.show()
- The result is shown in fig 9. the diagonal grouping of some pairs of attributes already show that there exists some correlation among the data.

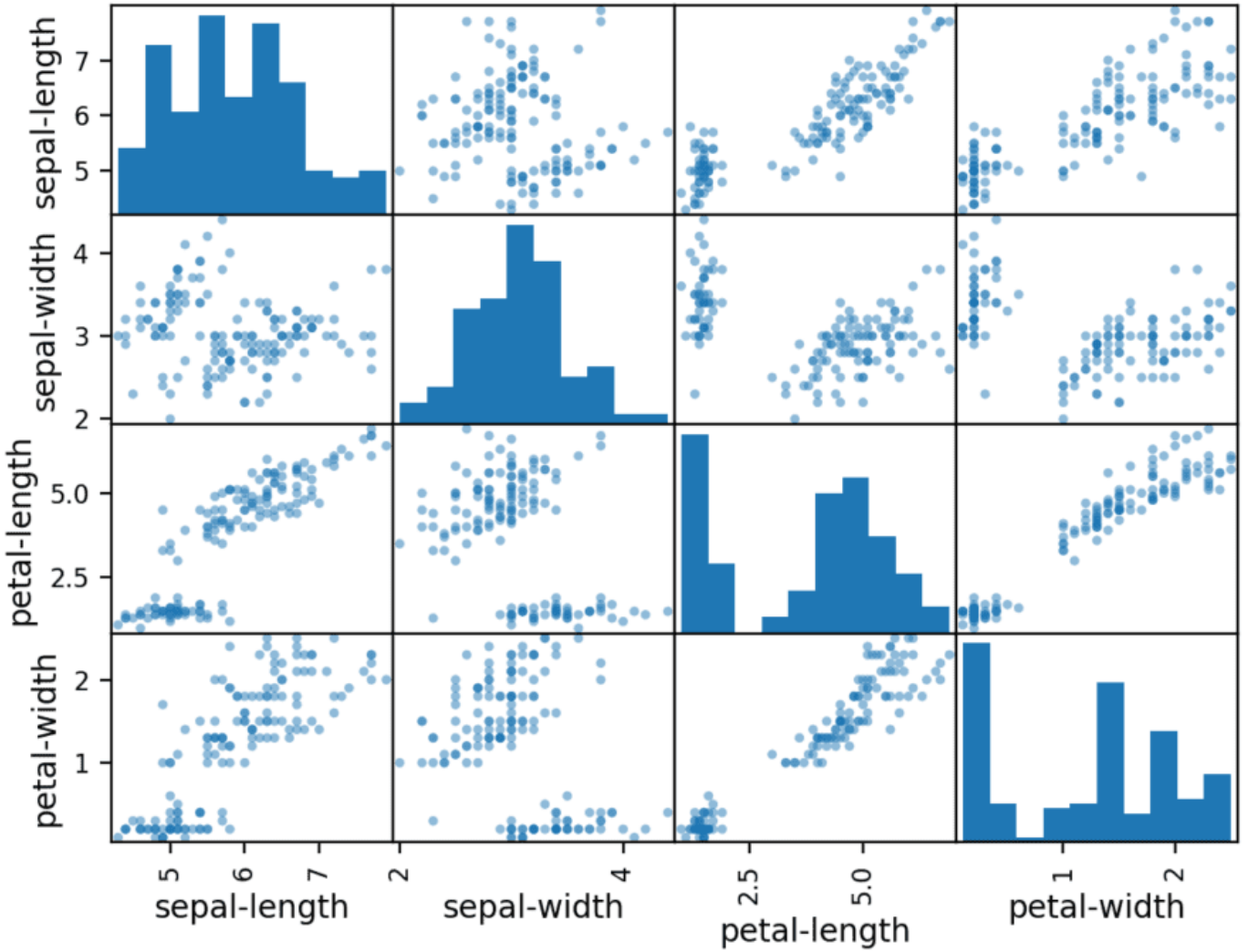

Fig 9 Scatter plot matrix of attributes (Brownlee, 2020)

5. Algorithm evaluation

- In this step the following shall be performed:
  a. Separate a validation dataset
  b. Perform testing using a 10 fold validation (k = 10)
  c. Build multiple different models for species prediction using the provided flower measurements in the dataset.
  d. Select the best model
  e. Compare the algorithms

(a) Separate a validation dataset

- The process of creating the folds has been described in an earlier lesson; to summarize we need to create a validation dataset separate from the one we shall use for training our model.
- To do so we need to split the data: a ratio of 80/20 is chosen, such that 80% of the data can be used for training while 20% will be used for validation.
- First we split the data:
- # Split-out validation dataset

```
array = dataset.values

X = array[:,0:4]

y = array[:,4]

X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20, random_state=1)
```

(b) Testing using k = 10 folds

- Using k = 10 folds involves the following:
- 9 datasets are trained while 1 is kept aside and used for training.
- In the next round a different set is removed and the remaining 9 are trained; the removed set then used for testing. This will go one till all the sets have been used for testing. In this case if our training examples are denoted by:
- $x_1, x_2, \ldots x_{10}$, then in round 1, x_{10} will be chosen for testing while the rest will be used for training
- In round 2, $x_{10}, x_1, x_2, \ldots x_8$ will be used for training and x_9 will be used for testing.
- The round robin approach will be used till all sets have each been used for both training and testing.
- Further according to Brownlee (2020), "We set the random seed via the *random_state* argument to a fixed number to ensure that each algorithm is evaluated on the same splits of the training dataset. The specific random seed does not matter"

(c) Build models

- Six different algorithms are tested in this tutorial. They are:
  - Logistic Regression (LR)
  - Linear Discriminant Analysis (LDA)
  - K-Nearest Neighbors (KNN).
  - Classification and Regression Trees (CART).
  - Gaussian Naive Bayes (NB).
  - Support Vector Machines (SVM)
- The code for building the models and evaluating them is presented next; a sample output is also presented.

(c) Build models

- ...
- # Spot Check Algorithms
- models = []
- models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
- models.append(('LDA', LinearDiscriminantAnalysis()))
- models.append(('KNN', KNeighborsClassifier()))
- models.append(('CART', DecisionTreeClassifier()))
- models.append(('NB', GaussianNB()))
- models.append(('SVM', SVC(gamma='auto')))
- # evaluate each model in turn
- results = []
- names = []
- for name, model in models:
- kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
- cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
- results.append(cv_results)
- names.append(name)
- print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

(d) Select best model

- #output
- LR: 0.960897 (0.052113)
- LDA: 0.973974 (0.040110)
- KNN: 0.957191 (0.043263)
- CART: 0.957191 (0.043263)
- NB: 0.948858 (0.056322)
- SVM: 0.983974 (0.032083)
- From the output above we can see that SVM (Support Vector Machine) has the best estimated accuracy at 98% and this is the model we choose.

(e) Comparing algorithms

- Since we used a 10 fold cross validation we can also plot a box and whisker plot for each of the distributions an compare them.
- The results are captured in fig 10 showing high accuracy for all the algorithms. The code used is:
- …
- # Compare Algorithms

  pyplot.boxplot(results, labels=names)

  pyplot.title('Algorithm Comparison')

  pyplot.show()

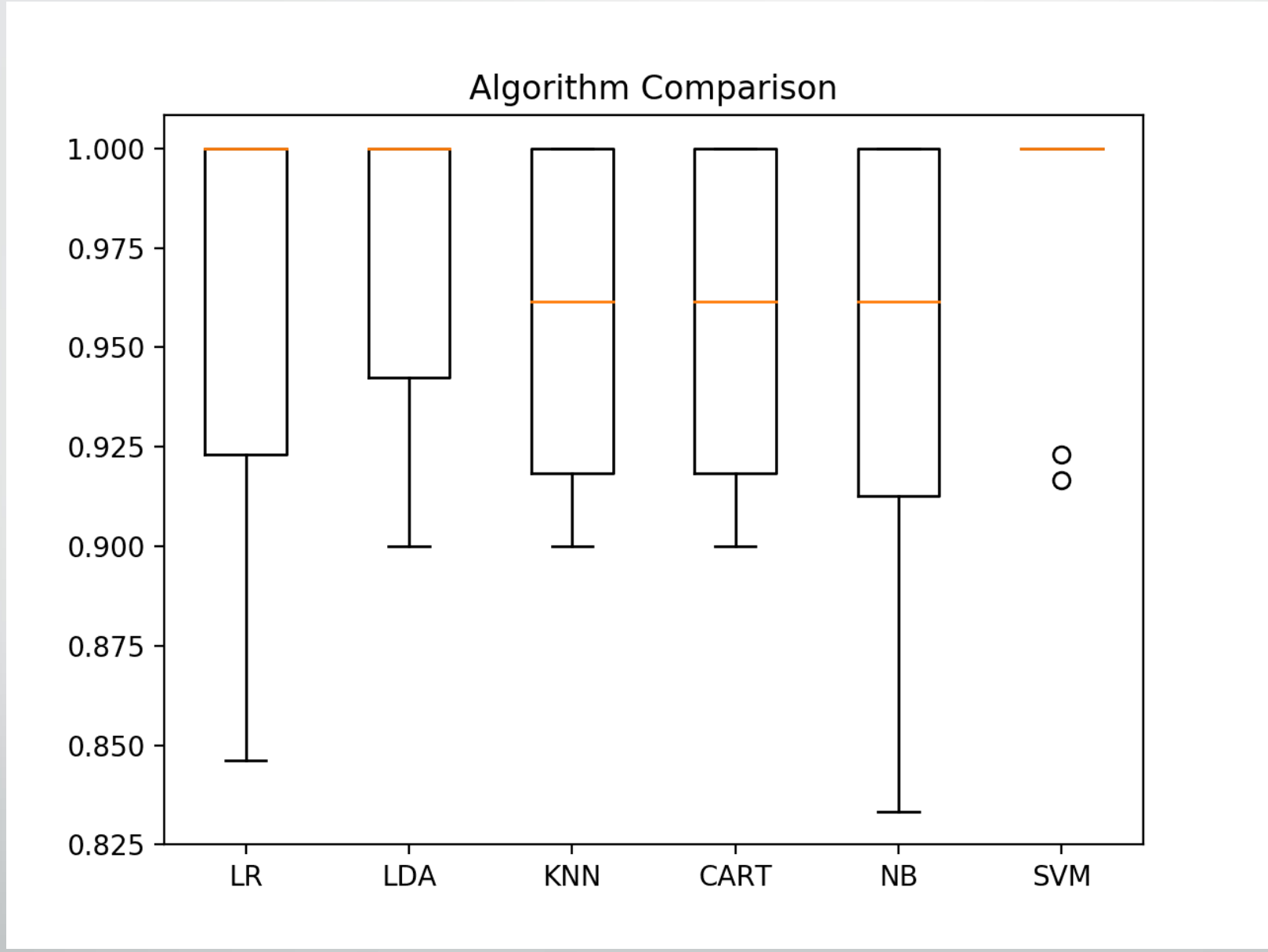

Fig 10 Algorithm comparison (Brownlee, 2020)

6. Making predictions

- So far we have used the 80% to train our data and to test it using the 10 fold cross validation method.
- The results show that SVM is perhaps the most accurate among them all.
- We would now like to confirm this by using our remaining 20% of the data.
- This was the one we split at the beginning of the training exercise so that we could use it for validation purposes at this point.
- Since this set was not used in training it will give us a true indication of how accurate our model is.
- We now fit the model on the whole dataset and make predictions on the validation set as follows:

6. Making predictions

- ...
- # Make predictions on validation dataset

  model = SVC(gamma='auto')

  model.fit(X_train, Y_train)

  predictions = model.predict(X_validation)

We can now check the accuracy of the predictions by comparing them to the expected results in the validation set, and also calculate classification accuracy, thus producing a confusion matrix and a classification report. The code to do so is as follows:

....

Evaluate predictions

print(accuracy_score(Y_validation, predictions))

print(confusion_matrix(Y_validation, predictions))

print(classification_report(Y_validation, predictions))

6. Making predictions

- The output is as follows:
- 0.9666666666666667
- [[11 0 0]
- [0 12 1]
- [0 0 6]]
- precision recall f1-score support

- Iris-setosa 1.00 1.00 1.00 11
- Iris-versicolor 1.00 0.92 0.96 13
- Iris-virginica 0.86 1.00 0.92 6

- accuracy 0.97 30
- macro avg 0.95 0.97 0.96 30
- weighted avg 0.97 0.97 0.97 30

Improving the results

- The steps covered in our hello world project covered all the project steps….except one.
- We described the accuracy of the algorithm for our model and found it was very high…97%
- Is this always the case? The answer of course is no.
- Therefore what options are available to machine learning results?
- In his paper on the same topic Brownlee (2020) offers three approaches:
- Algorithm tuning – this is done by fine tuning the configuration of the algorithms
- Ensembles – use ensemble methods such as bagging and boosting. These have been covered in this course
- Extreme feature engineering – expose more of the structure of the problem for the algorithm to learn. It is where "where the attribute decomposition and aggregation seen in data preparation is pushed to the limits."

Summary

- The three main data analysis packages are Numpy, Pandas, and Matplotlib.
- Numpy provides functionality to work with arrays from two dimensions to multidimensional arrays.
- Pandas works with data structures and is used for manipulations of tables and time series; the data structures are series and DataFrame.
- Matplotlib is the mathematics package used to produce all those nice diagrams that you see in so many publications.
- Every machine learning project follows a number of steps from inception to conclusion. These are: define the problem, prepare the data, evaluate algorithms, improve results, and present results.
- The options available to improve machine learning results are: algorithm tuning, ensembles, and extreme feature engineering.

References

- Brownlee, J. (2020, August 15). *How to improve machine learning results*. Machine Learning Mastery. Retrieved June 6, 2022, from https://machinelearningmastery.com/how-to-improve-machine-learning-results/
- Brownlee, J. (2020, August 19). *Your first machine learning project in python step-by-step*. Machine Learning Mastery. Retrieved June 6, 2022, from https://machinelearningmastery.com/machine-learning-in-python-step-by-step/
- Swamynathan, M. (2017). *Mastering machine learning with python in six steps a practical implementation guide to Predictive Data Analytics using python*. Apress, Springer Science+Business Media.