Contents

1 Anoop index: 1

2 experimental0 2

3 How Do I Get Started? 2

3.1 articles-read, worth rereading 2
3.2 notes 3
3.2.1 intro 3
3.2.2 some key word definitions: 3

4 `https://machinelearningmastery.com/python-machine-learning-mini-course/` 4

4.1 pre requisite `https://machinelearningmastery.com/gentle-introduction-to-the-bias-v`
4.2 1 - install 5
4.2.1 next time try using this tutorial `https://sourabhbajaj.com/mac-setup/Python/numpy.html` 5
4.2.2 make a new virtualenv 5
4.2.3 install `https://stackoverflow.com/questions/26319762/how-to-install-scipy-stack-with-pip-and-homebrew` 5
4.2.4 check if properly isntalled using .$_{version}$ _ after import 5
4.3 2 - python, pandas, numpy, mathplotlib veeery basics 6
4.3.1 python, also refer in-y-minutes file for future reference 6
4.3.2 numpy basics 7
4.3.3 matplotlib basics 11
4.3.4 Pandas -basics: 12
4.4 3 - Load csv 13
4.5 4 - use pandas.DataFrame helper functions to describe data with statistics 14
4.6 5 - Basic Data Visualization 15
4.7 6 - Preprocessing data 15
4.7.1 skip for now, come back later 17
4.8 7 - Resampling 17
4.8.1 Google Developers intro to ml 17

1 Anoop index:

How Do I Get Started? Applied Machine Learning Process Linear Algebra Statistical Methods Understand Machine Learning Algorithms Python Machine Learning (scikit-learn) Code Algorithm from Scratch (Python) Introduction to Time Series Forecasting (Python) XGBoost in Python (Stochastic Gradient Boosting) Deep Learning (Keras) Long Short-Term Memory (LSTM) Deep Learning for Natural Language Processing (NLP) Deep Learning for Time Series Forecasting

2 experimental0

chrome history suggestions based on a neural network read to get ideas for projects: `https://machinelearningmastery.com/self-study-machine-learning-projects/` maintain a ml daily blog? how to get computers to talk to each other using socket programming use free time with phone to study google maps of Bangalore like guttapalli buy toothbrush, bigger bulb for room sabji kitne ki hai app

3 How Do I Get Started?

3.1 articles-read, worth rereading

$$AppliedMachineLearningProcess$$

Intermediate: Python Ecosystem.

$$ Step 4: Practice on Datasets. Select datasets to work on and practice the process.

$$PracticeMachineLearningwithSmallIn-MemoryDatasets$$

Tour of Real-World Machine Learning Problems $$ Work on Machine Learning Problems That Matter To You

$$ Step 5: Build a Portfolio. Gather results and demonstrate your skills.

$$BuildaMachineLearningPortfolio$$

Get Paid To Apply Machine Learning $$ Machine Learning For Money

For more on this top-down approach, see:

$$TheMachineLearningMasteryMethod$$

Machine Learning for Programmers

Many of my students have used this approach to go on and do well in Kaggle competitions and get jobs as Machine Learning Engineers and Data Scientists.

3.2 notes

3.2.1 intro

sckit-learn is higher level than numpy & scipy machine learning is a subset of artificial intelligence artificial learning is a more consistent name for machine learning

3.2.2 some key word definitions:

Model: A machine learning model can be a mathematical representation of a real-world process. To generate a machine learning model you will need to provide training data to a machine learning algorithm to learn from.

Algorithm: Machine Learning algorithm is the hypothesis set that is taken at the beginning before the training starts with real-world data. When we say Linear Regression algorithm, it means a set of functions that define similar characteristics as defined by Linear Regression and from those set of functions we will choose one function that fits the most by the training data.

Training: While training for machine learning, you pass an algorithm with training data. The learning algorithm finds patterns in the training data such that the input parameters correspond to the target. The output of the training process is a machine learning model which you can then use to make predictions. This process is also called learning.

Regression: Regression techniques are used when the output is real-valued based on continuous variables. For example, any time series data. This technique involves fitting a line.

Classification: In classification, you will need to categorize data into predefined classes. For example, an email can either be spam or not spam.

Target: The target is whatever the output of the input variables. It could be the individual classes that the input variables maybe mapped to in case of a classification problem or the output value range in a regression problem. If the training set is considered then the target is the training output values that will be considered.

Feature: Features are individual independent variables that act as the input in your system. Prediction models use features to make predictions. New features can also be obtained from old features using a method known as feature engineering. More simply, you can consider one column of your data set to be one feature. Sometimes these are also called attributes. And the number of features are called dimensions.

Label: Labels are the final output. You can also consider the output classes to be the labels. When data scientists speak of labeled data, they mean groups of samples that have been tagged to one or more labels.

Overfitting: An important consideration in machine learning is how well the approximation of the target function that has been trained using training data, generalizes to new data. Generalization works best if the signal or the sample that is used as the training data has a high signal to noise ratio. If that is not the case, generalization would be poor and we will not get good predictions. A model is overfitting if it fits the training data too well and there is a poor generalization of new data.

Regularization: Regularization is the method to estimate a preferred complexity of the machine learning model so that the model generalizes and the over-fit/under-fit problem is avoided. This is done by adding a penalty on the different parameters of the model thereby reducing the freedom of the model.

Parameter and Hyper-Parameter: Parameters are configuration variables that can be thought to be internal to the model as they can be estimated from the training data. Algorithms have mechanisms to optimize parameters. On the other hand, hyperparameters cannot be estimated from the training data. Hyperparameters of a model are set and tuned depending on a combination of some heuristics and the experience and domain knowledge of the data scientist.

4 `https://machinelearningmastery.com/python-machine-learning-mini-`

14Lessons

4.1 pre requisite `https://machinelearningmastery.com/gentle-introduction-to-the-`

Bias Error Bias are the simplifying assumptions made by a model to make the target function easier to learn. Low Bias: Suggests less assumptions about the form of the target function. High-Bias: Suggests more assumptions about the form of the target function.

Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines. Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Variance Error Variance is the amount that the estimate of the target function will change if different training data was used.

Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression. Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Fig 1. bulls-eye visualise `http://scott.fortmann-roe.com/docs/BiasVariance.html`

4.2 1 - install

4.2.1 next time try using this tutorial `https://sourabhbajaj.com/mac-setup/Python/numpy.html`

4.2.2 make a new virtualenv

```
pwd
```

Use :session: property to speed up org-babel when possible to use the same session

```
source ~/.bashrc
#mkvirtualenv mlm
workon mlm
which python
```

```
(pyvenv-workon "mlm2")
```

4.2.3 install `https://stackoverflow.com/questions/26319762/how-to-install-scipy-stack-w`

pip install numpy brew install gcc pip install scipy brew install freetype pip install matplotlib pip install nose pip install pandas pip install sympy pip install ipython[all] brew install pyqt brew install qt brew install sip #after this edit the 2 scripts

4.2.4 check if properly isntalled using .$\_{version}$ _ after import

using python snippets inside orgmode `https://orgmode.org/worg/org-contrib/babel/languages/ob-doc-python.html` installed python-mode from package-list-packages for emacs

```
(pyvenv-workon "mlm")
```

Fixed some errors using: pip install nose pyparsing python-dateutil pep8

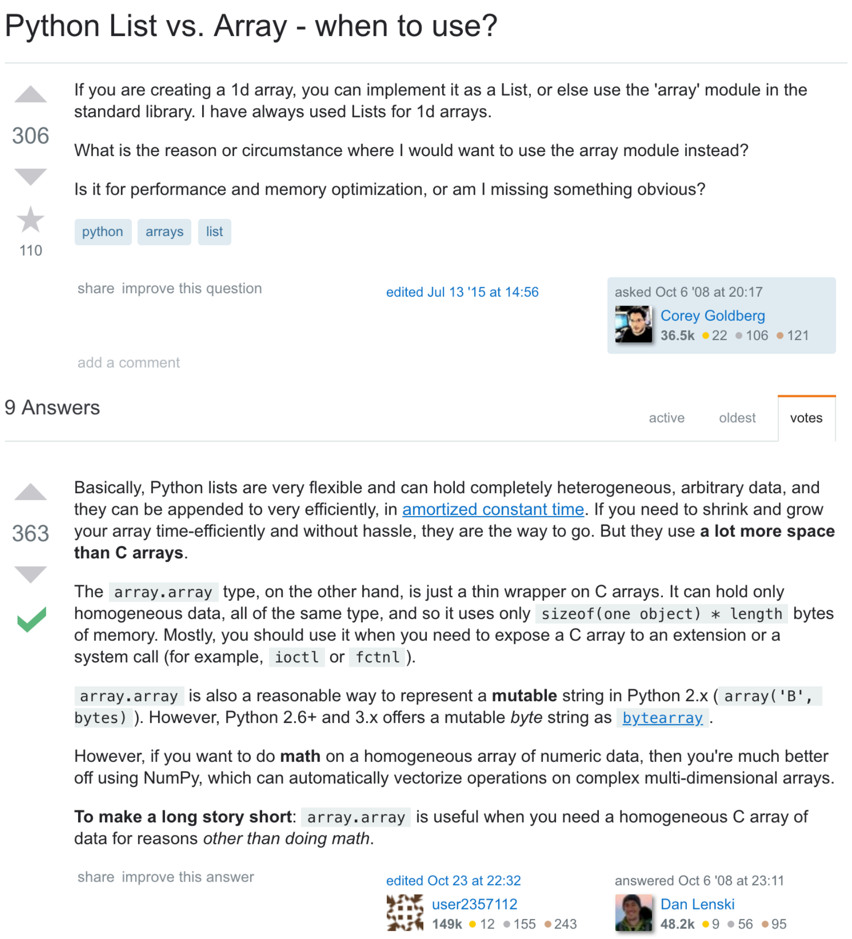
```
import sys
import scipy
import numpy
import matplotlib
import pandas
import sklearn

print(f'python: {sys.version}')
print(f'scipy: {scipy.__version__}')
print(f'numpy: {numpy.__version__}')
print(f'matplotlib {matplotlib.__version__}')
print(f'pandas {pandas.__version__}')
print(f'sklearn {sklearn.__version__}')
```

4.3 2 - python, pandas, numpy, mathplotlib veeery basics

4.3.1 python, also refer in-y-minutes file for future reference

```
if 1>2:
    print("wtf")
else:
    print("ok")

try:
    # Use "raise" to raise an error
    raise IndexError("This is an index error")
except IndexError as e:
    print("its indexerror")
    pass                 # Pass is just a no-op. Usually you would do recovery here.
except (TypeError, NameError):
    print("its typeerror or nameerror")
    pass                 # Multiple exceptions can be handled together, if required.
else:                    # Optional clause to the try/except block. Must follow all exc
    print("All good!")   # Runs only if the code in try raises no exceptions
finally:                 #  Execute under all circumstances
    print("We can clean up resources here")

def accepts_variable_number_of_arguments(*args):
    print(type(args))
    print(args)

accepts_variable_number_of_arguments(1,2,3)

def accepts_variable_number_of_keyword_arguments(**kwargs):
    print(type(kwargs))
    print(kwargs)

accepts_variable_number_of_keyword_arguments(name="anoop", work="code")
```

4.3.2 numpy basics



Python List vs. Array - when to use?

306

If you are creating a 1d array, you can implement it as a List, or else use the 'array' module in the standard library. I have always used Lists for 1d arrays.

What is the reason or circumstance where I would want to use the array module instead?

Is it for performance and memory optimization, or am I missing something obvious?

python arrays list

110

share improve this question

edited Jul 13 '15 at 14:56

asked Oct 6 '08 at 20:17
Corey Goldberg
36.5k ●22 ●106 ●121

add a comment

9 Answers

active oldest votes

363

Basically, Python lists are very flexible and can hold completely heterogeneous, arbitrary data, and they can be appended to very efficiently, in amortized constant time. If you need to shrink and grow your array time-efficiently and without hassle, they are the way to go. But they use **a lot more space than C arrays**.

The `array.array` type, on the other hand, is just a thin wrapper on C arrays. It can hold only homogeneous data, all of the same type, and so it uses only `sizeof(one object) * length` bytes of memory. Mostly, you should use it when you need to expose a C array to an extension or a system call (for example, `ioctl` or `fctnl`).

`array.array` is also a reasonable way to represent a **mutable** string in Python 2.x (`array('B', bytes)`). However, Python 2.6+ and 3.x offers a mutable *byte* string as `bytearray` .

However, if you want to do **math** on a homogeneous array of numeric data, then you're much better off using NumPy, which can automatically vectorize operations on complex multi-dimensional arrays.

To make a long story short: `array.array` is useful when you need a homogeneous C array of data for reasons *other than doing math*.

share improve this answer

edited Oct 23 at 22:32
user2357112
149k ●12 ●155 ●243

answered Oct 6 '08 at 23:11
Dan Lenski
48.2k ●9 ●56 ●95

numpy official tutorial https://docs.scipy.org/doc/numpy-1.15.0/user/quickstart.html

```
import numpy as np
a = np.arange(15).reshape(3,5)
print(a)
```

```python
a.shape
a.ndim
a.dtype.name
#dir(a)
a.size
type(a)
b = np.array([6,7,8])
b
type(b)

np.zeros([2,3])
np.arange(15)
np.linspace(0,9, 19)

from numpy import pi
x = np.linspace(0, 2*pi, 5)
np.sin(x)
#2 decimal places
np.around(np.sin(x), decimals=2)

A = np.array([[1,2],[3,4]])
I = np.array([[1,0],[0,1]])
elementwise = A * I
matrix_product = A @ I
print(elementwise, "\n", matrix_product)

a = np.ones(3, dtype=np.int32)
b = np.linspace(0, 1, 3)
c = a + b
print(a,b,c)
c.dtype.name
c*1j
np.ones(1)
my_e = np.exp(np.ones(1))
from numpy import e
e
print(e, my_e)
d = np.exp(c*1j)
d.dtype
# exp, sin etc are called numpy universal functions
```

```
  # multidimensional array
c = np.array([
    [
^^I[  0,  1,  2],
^^I[ 10, 12, 13]
    ],
    [
^^I[100,101,102],
^^I[110,112,113]
    ]
])
c.shape
# Visualize0 2,2,3 as you traverse from the topmost bracket to the inner ones
for i in c.flat:
    print(i, end=" // ")
print("\n")
id(c) #id is unique identifier of an object in python
```

axis in numpy

file:~/ml/flipshope/screenshots0/Screenshot 2018-12-11 at 6.06.43 PM.png

matplotlib python is not installed as a framework error, solution: `https://stackoverflow.com/questions/34977388/matplotlib-runtimeerror-python-is-not-installed`

Above is a hacky solution I need to switch away from virtualenv & virtualenvwrapper and move to venv entirely Also, reddit recommends to avoid the pyenv or any wrapper around venv && strongly recommends to use venv directly venv ships by default with python >= 3.3 `https://matplotlib.org/faq/osx_framework.html` `https://news.ycombinator.com/item?id=18612590` `https://news.ycombinator.com/item?id=18247512` :) andybak 54 days ago [-] In case this scares any new users, I've used nothing more than pip and virtualenv for several years with no issues of note.

```
import numpy as np
import matplotlib.pyplot as plt
#import matplotlib
#matplotlib.use('TkAgg')
#import matplotlib.pyplot as plt

def mandelbrot( h,w, maxit=20 ):
    """Returns an image of the Mandelbrot fractal of size (h,w)."""
    y,x = np.ogrid[ -1.4:1.4:h*1j, -2:0.8:w*1j ]
    c = x+y*1j
    z = c
    divtime = maxit + np.zeros(z.shape, dtype=int)

    for i in range(maxit):
^^Iz = z**2 + c
^^Idiverge = z*np.conj(z) > 2**2            # who is diverging
^^Idiv_now = diverge & (divtime==maxit)  # who is diverging now
^^Idivtime[div_now] = i                  # note when
^^Iz[diverge] = 2                        # avoid diverging too much

    return divtime
plt.imshow(mandelbrot(400,400))
plt.show()

import sys
print(sys.path)
```

real-python tutorial `https://realpython.com/numpy-array-programming/`

When it comes to computation, there are really three concepts that lend NumPy its power: Vectorization Broadcasting Indexing

```
import numpy as np

arr = np.arange(36).reshape(3,4,3)
arr

"""
visualize0:-

00 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35

00 01 02
03 04 05
06 07 08
09 10 11
```

```
[#3 items
[#4 items
[]
[]
[]
[]
]...
]
"""
a = np.array([2,3,4])
b = 2
b_broadcasted = np.broadcast(a,b)
print(list(b_broadcasted))
# rest of this tutorial seemed a bit advanced, skip for now, come back later
```

Note Pandas is a library built on top of NumPy

todo: switch to jupyter notebook instead of emacs: https://github.com/millejoh/emacs-ipython-notebook http://millejoh.github.io/emacs-ipython-notebook/ https://www.youtube.com/watch?v=wtVF5cMhBjg https://news.ycombinator.com/item?id=9728143 https://github.com/gregsexton/ob-ipython

4.3.3 matplotlib basics

matplotlib, is written in pure Python and is heavily dependent on NumPy

Matplotlib is conceptually divided into three parts: pylab interface (similar to MATLAB) pylab tutorial, this is the matplotlib.pyplot import Matplotlib frontend or API artist tutorial backends drawing devices or renderers

Lets learn pyplot https://matplotlib.org/users/pyplot_tutorial.html#pyplot-tutorial

venv basics to switch from virtualenv

python3 -m venv mlm2 source mlm2/bin/activate

pyvenv-deactivate

```
import matplotlib
#matplotlib.use('MacOSX')
import matplotlib.pyplot as plt

plt.plot([0, 1, 4, 9, 16])
#plt.show()
```

```
plt.plot([0, 1, 4, 9, 16], 'ro')
#plt.show()

plt.plot([0.5], 'y.')
plt.show()
```

4.3.4 Pandas -basics:

Todo:https://pandas.pydata.org/pandas-docs/stable/dsintro.html#dsintro
Todo:Official beginner tutorial: https://pandas.pydata.org/pandas-docs/stable/10min.html Todo: Intermediate - Julia Evans - https://jvns.ca/blog/2013/12/22/cooking-with-pandas/ "take a real dataset or three, play around with it, and learn how to use pandas along the way."

Panda Series & DataFrames: https://medium.freecodecamp.org/series-and-dataframe-in-pyt

```
#Series and DataFrames
import pandas as pd
x1 = pd.Series([6,3,4,6])
x = pd.Series([6,3,4,6], index=['a','b','c','d'])
x
y = pd.Series(3, index=['a', 'b', 'c', 'd'])
y

#DataFrames
import numpy as np
dates = pd.date_range('20181201', periods = 8)

my_narray = np.random.randn(8,3)
list('ABC')

df = pd.DataFrame(index = dates, data = my_narray, columns = ['A','B','C'])
df_absolute = df.apply(abs)
```

So pandas is kinda like an excel sheet0

```
import numpy as np

np.arange(4)
ma = np.arange(4).reshape((2,2))

import pandas
```

```
p = pandas.DataFrame(ma)

print(ma[1,0])
print(p[1])
print(p[1][0] == ma[1][0])
print(p.shape)
```

4.4 3 - Load csv

https://realpython.com/python-csv/ https://github.com/jbrownlee/Datasets

Work with csv using python's csv module

```
import csv

with open('iris.csv') as csv_file:
    """
    for line in csv_file:
^^Iprint(line)
^^Ipass
    """
    csv_reader = csv.reader(csv_file)
    for line in csv_reader:
^^I#print(line)
^^Ipass

my_fieldnames = ("sepal_length", "sepal_width", "petal_length", "petal_width", "class"

with open('iris.csv') as csv_file:
    csv_dict_reader = csv.DictReader(csv_file, fieldnames=my_fieldnames)
    for line in csv_dict_reader:
^^I#print(line)
^^Ipass

with open('test_writeout.csv', mode='w') as out_file:
    csv_writer = csv.writer(out_file)
    csv_writer.writerow(["row", "1"])
    csv_writer.writerow(["row", "2"])
    #
    csv_dict_writer = csv.DictWriter(out_file, fieldnames = my_fieldnames)
    csv_dict_writer.writeheader()
    csv_dict_writer.writerow({"sepal_length": 1, "sepal_width": 2, "petal_length": 3,
```

Work with csv using numpy

```
import numpy as np

with open("numpy_loadtxt_input.txt") as input_file:
    """
    for line in input_file:
^^Iprint(line)
    """
    my_nparray = np.loadtxt(input_file, delimiter=" ")
    print(my_nparray)
    print(my_nparray.dtype)
```

Work with csv usign pandas

```
import pandas
df = pandas.read_csv('pandas_read_csv.csv')
print(df)

df2 = pandas.read_csv('pandas_read_csv.csv', parse_dates=['Hire Date'], index_col='Nam
print(df2)

my_col_names = ("Name", "Hired_on", "Salary", "sick_days_remaining")
df3 = pandas.read_csv('pandas_read_csv.csv', header=None, names=my_col_names,
parse_dates=['Hired_on'])
print(df3)

df3.to_csv('pandas_to_csv.csv')
```

4.5 4 - use pandas.DataFrame helper functions to describe data with statistics

```
# Scatter Plot Matrix
import matplotlib.pyplot as plt
import pandas

my_col_names = ("num_preg", "plasma_glucose", "blood_pressure", "triceps_skin_thicknes
df = pandas.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima
#print(df)

from pandas.plotting import scatter_matrix

scatter_matrix(df)
plt.show()

df.corr()
```

4.6 5 - Basic Data Visualization

Using pandas and matplotlib together

```
# Scatter Plot Matrix
import matplotlib.pyplot as plt
import pandas

my_fieldnames = ("sepal_length", "sepal_width", "petal_length", "petal_width", "class"
data = pandas.read_csv('iris.csv', names=my_fieldnames)
#print(data)

se = data.loc[data["class"] == "Iris-setosa"]
ve = data.loc[data["class"] == "Iris-versicolor"]
vi = data.loc[data["class"] == "Iris-virginica"]

#se.hist()
#ve.hist()
#vi.hist()

se.plot(kind="box")

from pandas.plotting import scatter_matrix
#scatter_matrix(se)

plt.savefig("img/lesson5.png")
return "img/lesson5.png"
```

4.7 6 - Preprocessing data

Standardize numerical data (e.g. mean of 0 and standard deviation of 1) using the scale and center options.

Simple example:

```
import numpy
narray = numpy.arange(0,4).reshape(2,2)

import pandas
df = pandas.DataFrame(narray, columns=("c1", "c2"))

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df) #calculates and store mean & standard-deviation for each column

print(scaler.transform(df))
type(scaler.transform(df))
df_standardized = pandas.DataFrame(scaler.transform(df))
```

Complex example:

```
import matplotlib.pyplot as plt
import pandas

#df is my short for DataFrame
my_col_names = ("num_preg", "plasma_glucose", "blood_pressure", "triceps_skin_thicknes
df = pandas.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-
#print(df)

import numpy
array = df.values

X = array[:, :-1]
Y = array[:, -1:]

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
X_standardized = scaler.transform(X)

#printout
numpy.set_printoptions(precision=2)
print(X_standardized)

df_standardized = pandas.DataFrame(X_standardized)
df_standardized.describe() #you can see that mean=0, sdev=1 for each column
```

4.7.1 skip for now, come back later

Normalize numerical data (e.g. to a range of 0-1) using the range option. Explore more advanced feature engineering such as Binarizing.

4.8 7 - Resampling

statistical methods called resampling methods are used to split your training dataset up into subsets, some are used to train the model and others are held back and used to estimate the accuracy of the model on unseen data.

4.8.1 Google Developers intro to ml

complete google developers course, its a pre-requisite for this lesson as per me

1. 1 hello apple or orange using decision tree

   ```
   from sklearn import tree
   features_original = [[140, "smooth"], [130, "smooth"], [150, "bumpy"], [170, "bump
   labels_original = ["apple", "apple", "orange", "orange"]

   #smooth=1 // bumpy=0
   #orange=1 // apple=0
   features = [[140, 1], [130, 1], [150, 0], [170, 0]] #weight, texture of fruit
   labels = [0, 0, 1, 1]

   #train a classifier:
   clf = tree.DecisionTreeClassifier() #instantiate an empty box of rules
   clf = clf.fit(features, labels) #learning algorithm fills the above box with rule

   #print(clf.predict([[150,0]]))
   print(clf.predict([[200,0]]))
   ```

2. 2 Decision Tree visualization

   ```
   from sklearn.datasets import load_iris
   from sklearn import tree
   iris = load_iris()
   print(dir(iris))

   print(iris.feature_names) #in this example data_names is more suitable
   ```

```
print(iris.data[0])

print(iris.target_names)
print(iris.target)

#Resampling:
test_ids = [0,50,100]

import numpy as np

#training data
train_target = np.delete(iris.target, test_ids)
train_data = np.delete(iris.data, test_ids, axis=0)

#testing data
test_target = iris.target[test_ids]
test_data = iris.data[test_ids]

clf = tree.DecisionTreeClassifier()
clf.fit(train_data, train_target)

predicted_target = clf.predict(test_data)
print(f"Reality: {test_data} features is {test_target}")
print(f"Prediction: {test_data} has been predicted as {predicted_target}")

#Visualize: https://medium.com/@rnbrown/creating-and-visualizing-decision-trees-w
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus

dot_data = StringIO() #StringIO() behaves like a file
export_graphviz(clf, out_file=dot_data, feature_names=iris.feature_names, class_na
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png("img/iris.png")
return "img/iris.png"
```