COMP4211-Tutorial 2: Python Basics and Perceptron Algorithm

Weiyu CHEN, wchenbx@connect.ust.hk

Part I: Python Basics

This part of notebook credits by cs228.

Python is a great general-purpose programming language on its own, but with the help of a few popular libraries (numpy, scipy, matplotlib) it becomes a powerful environment for scientific computing.

Some of you may have previous knowledge in Matlab, in which case we also recommend the numpy for Matlab users page.

Hello World

In [1]:

```
print('Hello world from COMP4211')
```

```
Hello world from COMP4211
```

Types & Operations

Numbers

In [2]:

```
x = 114514
print(x, type(x))
y = 1919810.
print(y, type(y))
```

```
114514 <class 'int'>
1919810.0 <class 'float'>
```

In [3]:

```
print(x + 1919810)    # Addition;
print(x - 1919810)    # Subtraction;
print(x * 1919810)    # Multiplication;
print(x // 1919810)   # Integer Division;
print(x / 1919810)    # Division;
print(x **  20)        # Exponentiation;
# matrix multiplication: @
```

```
2034324
-1805296
219845122340
0
0.05964861106046952
150373635229950886663758295582434037200633466483715311561241291591000359303505066112699986840912920576
```

In [4]:

```
x += 1
print(x)
x *= 2
print(x)
```

```
114515
229030
```

In [5]:

```
a = 2
b = 3
c = a / b
d = a // b
print(a, type(a))
print(b, type(b))
print(c, type(c))
print(d, type(d))
```

```
2 <class 'int'>
3 <class 'int'>
0.6666666666666666 <class 'float'>
0 <class 'int'>
```

Booleans

In [6]:

```
t, f = True, False
print(t, type(t), f, type(f))
```

```
True <class 'bool'> False <class 'bool'>
```

In [7]:

```
print(t and f) # Logical AND;
print(t or f)  # Logical OR;
print(not t)   # Logical NOT;
print(t != f)  # Logical XOR;
```

```
False
True
False
True
```

Strings

In [10]:

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello, type(hello), len(hello))
```

```
hello <class 'str'> 5
```

In [11]:

```
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
```

```
hello world
```

You can find a list of all string methods in the documentation.

Containers

Python includes several built-in container types: lists, dictionaries, sets, and tuples.

Lists

A list is the Python equivalent of an array, but is resizeable and can contain elements of different types:

In [12]:

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])
print(xs[-1])     # Negative indices count from the end of the list; print:
```

```
[3, 1, 2] 2
2
```

In [13]:

```
xs[2] = 'foo'    # Lists can contain elements of different types
print(xs)
```

```
[3, 1, 'foo']
```

In [14]:

```
xs.append('bar') # Add a new element to the end of the list
print(xs)
```

```
[3, 1, 'foo', 'bar']
```

In [15]:

```
x = xs.pop()     # Remove and return the last element of the list
print(x, xs)
```

```
bar [3, 1, 'foo']
```

As usual, you can find all the gory details about lists in the documentation.

You can **loop** over the elements of a list like this:

In [16]:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

Packages

Packages in python are some programs pre-written by others for us to use in a few lines. To use packages, we need to first install them by `conda` or `pip` as taught in the last tutorial. After installing them, we can simply `import` them, then we can use those powerful programs from giants.

For example, if we want to use `sci-learn`, we can simply `import skearn`.

In [17]:

```
import sklearn
```

Then we can use any functions or classes from `sci-learn`. For example, we want to check the version of the installed `ski-learn`.

In [18]:

```
print(sklearn.__version__)
```

```
0.24.2
```

We can import only parts of the packages, and also give them alias for convenience.

In [17]:

```
from sklearn import __version__ as sklearn_version
print(sklearn_version)
```

```
0.24.2
```

Part II: Perceptron Algorithm in Python

The perceptron is a simple supervised machine learning algorithm and one of the earliest **neural network** architectures. It was introduced by Rosenblatt in the late 1950s. A perceptron represents a **binary linear classifier** that maps a set of training examples (of d dimensional input vectors) onto binary output values using a $d-1$ dimensional hyperplane.

The perceptron as follows.

Given:

- dataset $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$
- with $\boldsymbol{x}^{(i)}$ being a d-dimensional vector $\boldsymbol{x}^{i} = (x_1^{(i)}, \ldots, x_d^{(i)})$
- $y^{(i)}$ being a binary target variable, $y^{(i)} \in \{0, 1\}$

The perceptron is a very simple neural network:

- it has a real-valued weight vector $\boldsymbol{w} = (w^{(1)}, \ldots, w^{(d)})$
- it has a real-valued bias b
- it uses the Heaviside step function as its activation function

---

Step 1: Initialize the weight vector and bias with zeros (or small random values).

Step 2: Compute a linear combination of the input features and weights. This can be done in one step for all training examples, using vectorization and broadcasting:

$$\boldsymbol{a} = \boldsymbol{X} \cdot \boldsymbol{w} + b$$

where $\boldsymbol{X}$ is a matrix of shape $(n_{samples}, n_{features})$ that holds all training examples, and $\cdot$ denotes the dot product.

Step 3: Apply the Heaviside function, which returns binary values:

$$\hat{y}^{(i)} = 1 \, \mathrm{if} \, a^{(i)} \geq 0, \, \mathrm{else} \, 0$$

Step 4: Update the weights and bias, which could be done by sklearn automatically.

Import needed packages and set the random seed manually to reproduce the result.

In [29]:

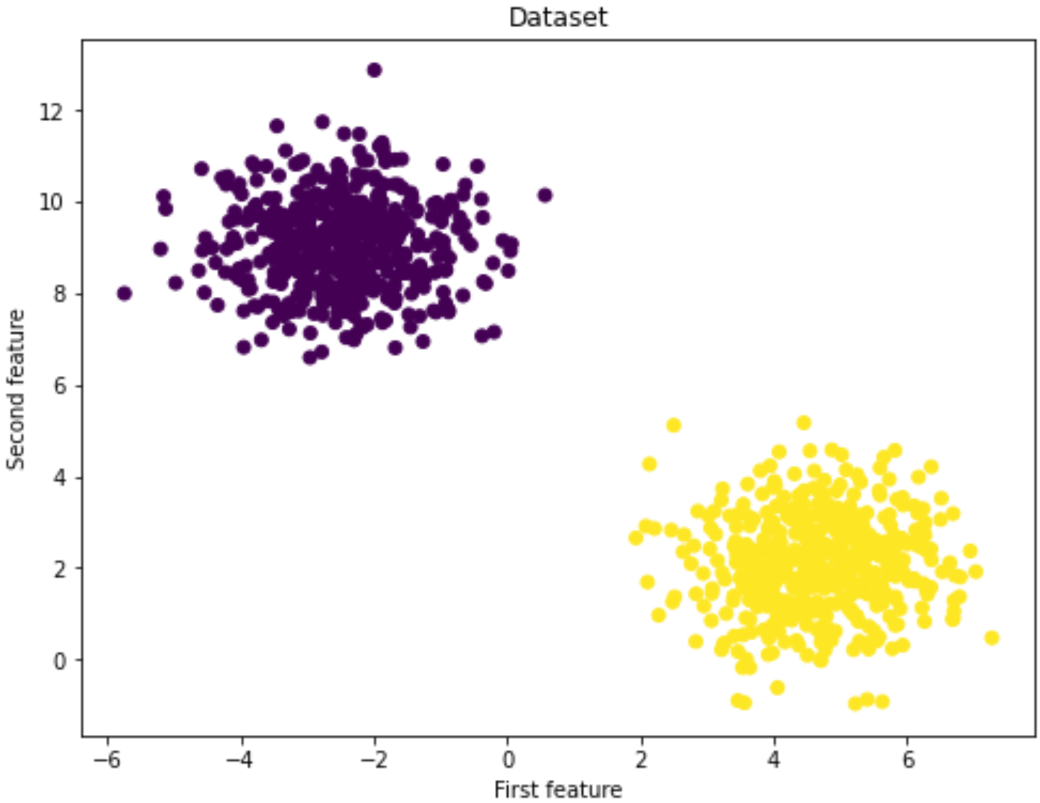
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron

np.random.seed(42)
```

Make the dataset in 2D.

In [30]:

```
X, y = make_blobs(n_samples=1000, centers=2)
fig = plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=y)
plt.title("Dataset")
plt.xlabel("First feature")
plt.ylabel("Second feature")
plt.show()
```



Split the dataset into training set and testing set.

In [31]:

```
y_true = y[:, np.newaxis]

X_train, X_test, y_train, y_test = train_test_split(X, y_true)
y_train = y_train.squeeze()
y_test = y_test.squeeze()

print(f'Shape X_train: {X_train.shape}')
print(f'Shape y_train: {y_train.shape}')
print(f'Shape X_test: {X_test.shape}')
print(f'Shape y_test: {y_test.shape}')
```

```
Shape X_train: (750, 2)
Shape y_train: (750,)
Shape X_test: (250, 2)
Shape y_test: (250,)
```

In [32]:

```
p = Perceptron(max_iter=10, verbose=1, random_state=1)
```

In [33]:

```
p.fit(X_train, y_train)
```

```
-- Epoch 1
Norm: 12.73, NNZs: 2, Bias: 1.000000, T: 750, Avg. loss: 0.007739
Total training time: 0.00 seconds.
-- Epoch 2
Norm: 12.73, NNZs: 2, Bias: 1.000000, T: 1500, Avg. loss: 0.000000
Total training time: 0.00 seconds.
-- Epoch 3
Norm: 12.73, NNZs: 2, Bias: 1.000000, T: 2250, Avg. loss: 0.000000
Total training time: 0.00 seconds.
-- Epoch 4
Norm: 12.73, NNZs: 2, Bias: 1.000000, T: 3000, Avg. loss: 0.000000
Total training time: 0.00 seconds.
-- Epoch 5
Norm: 12.73, NNZs: 2, Bias: 1.000000, T: 3750, Avg. loss: 0.000000
Total training time: 0.00 seconds.
-- Epoch 6
Norm: 12.73, NNZs: 2, Bias: 1.000000, T: 4500, Avg. loss: 0.000000
Total training time: 0.00 seconds.
-- Epoch 7
Norm: 12.73, NNZs: 2, Bias: 1.000000, T: 5250, Avg. loss: 0.000000
Total training time: 0.00 seconds.
Convergence after 7 epochs took 0.00 seconds
```

Out[33]:

```
Perceptron(max_iter=10, random_state=1, verbose=1)
```

In [34]:

```
y_p_train = p.predict(X_train)
y_p_test = p.predict(X_test)

print(f"training accuracy: {p.score(X_train, y_train) * 100}%")
print(f"test accuracy: {p.score(X_test, y_test) * 100}%")
```

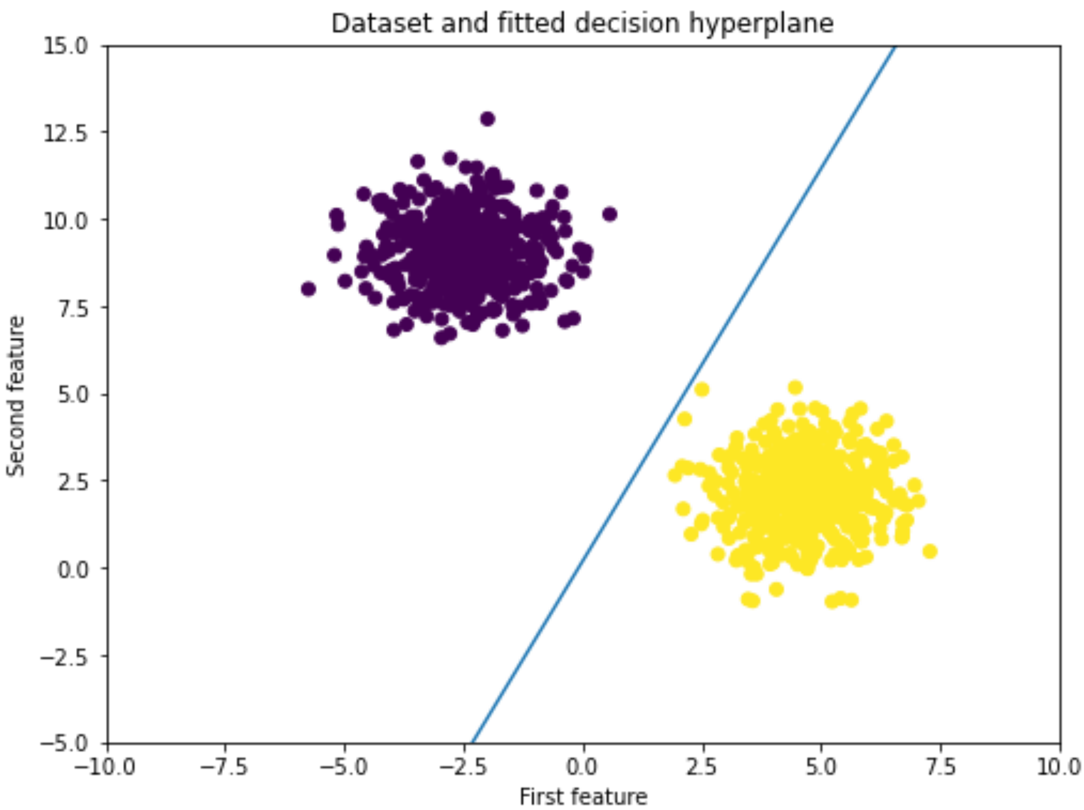
```
training accuracy: 100.0%
test accuracy: 100.0%
```

Plot the result on all data and the dicision hyperplane:

In [37]:

```
# weights learnt
weights = p.coef_[0]
offset = p.intercept_[0]
print(f"weights={weights}, offset={offset}")
w1 = weights[0]
w2 = weights[1]
b = offset
slope = -w1/w2
intercept = -b/w2
Xs = np.linspace(-10,10,10)
ys = slope * Xs + intercept
fig = plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=y)
plt.plot(Xs, ys, '-')
plt.title("Dataset and fitted decision hyperplane")
plt.xlabel("First feature")
plt.ylabel("Second feature")
plt.xlim(-10, 10)
plt.ylim(-5, 15)
plt.show()
```

```
weights=[11.63324772 -5.16961725], offset=1.0
```



For details, you may refer to the source code of `sklearn` or the perceptron algorithm demo without using `sklearn` .