Bootcamp Python

Day00 - Mathematical Delights

Starting by recoding some usefull functions and formulas used later during the week.

The idea is to help the students to 'tame' the mathematical jargon. They will have to simply implement the functions following the formulas without knowing what it is about. Then magically these already 'tamed' material will be used in context and understood without having to fight with its abstract notation.

Notions of the day

Sum, mean, variance, standard deviation, vectors and matrices operations.

General rules

- The version of Python to use is 3.7, you can check the version of Python with the following command: python -V
- The norm: during this bootcamp you will follow the Pep8 standards https://www.python.org/dev/peps/pep-0008/
- The function eval is never allowed.
- The exercices are ordered from the easiest to the hardest.
- Your exercices are going to be evaluated by someone else so make sure that variables and functions names are appropriated.
- Your man is internet.
- You can also ask question in the dedicated channel in Slack: 42-ai.slack.com.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on Github: https://github.com/42-AI/bootcamp_machine-learning/issues.

Helper

Ensure that you have the right Python version.

```
> which python
/goinfre/miniconda/bin/python
> python -V
Python 3.7.*
> which pip
/goinfre/miniconda/bin/pip
```

Exercise 00 - Sum

Exercise 01 - Mean

Exercise 02 - Variance

Exercise 03 - Standard deviation

Exercise 04 - Dot product

Exercise 05 - Matrix-vector multiplication

Exercise 06 - Matrix-matrix multiplication

Exercise 07 - Mean Squared Error - iterative

Exercise 08 - Mean Squared Error - vectorized

Exercise 09 - Mean Squared Error as linear cost function - iterative

Exercise 10 - Mean Squared Error as linear cost function - vectorized

Exercise 11 - Linear Gradient - iterative

Exercise 12 - Linear Gradient - vectorized

Exercise 00 - Sum

Turnin directory :	ex00
Files to turn in :	sum.py
Forbidden function :	*.sum()
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$\sum_{i=1}^{n} f(x_i)$$

where:

- x_i is the ith component of the vector x
- f is a function applied to x_i. In other words f is a function applied element-wise to the vector x.

Instructions:
In the sum.py file create the following function as per the instructions given below:

```
def sum_(x, f):
    """Computes the sum of a non-empty numpy.ndarray onto wich a function is
applied element-wise, using a for-loop.
    Args:
     x: has to be an numpy.ndarray, a vector.
     f: has to be a function, a function to apply element-wise to the
vector.
    Returns:
     The sum as a float.
     None if x is an empty numpy.ndarray or if f is not a valid function.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21])
>>> sum_(X, lambda x: x)
7
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21])
>>> sum_(X, lambda x: x**2)
949
```

Exercise 01 - Mean

Turnin directory :	ex01
Files to turn in :	mean.py
Forbidden function :	*.sum(), np.mean()
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} f(x_i)$$

where:

- mu is the mathematical notation of the mean, which is a real number
- x_i is the ith component of the vector x
- m is the number of elements of x

Instructions:
In the mean.py file create the following function as per the instructions given below:

```
def mean(x):
    """Computes the mean of a non-empty numpy.ndarray, using a for-loop.
    Args:
     x: has to be an numpy.ndarray, a vector.
    Returns:
     The mean as a float.
     None if x is an empty numpy.ndarray or if f is not a valid function.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21])
>>> mean(X)
1.0
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21])
>>> mean(X ** 2)
135.57142857142858
```

Exercise 02 - Variance

Turnin directory :	ex02
Files to turn in :	variance.py
Forbidden function :	*.sum(), np.mean(), np.var()
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \frac{1}{m}\sum_{i=1}^{m}x_i)^2$$

where:

- σ^2 is the mathematical notation of the variance, which is a real number
- x is a vector of dimension m * 1
- x_i is the ith component of the vector x
- m is the number of components in x

Do not worry! You have already encounter half of this ugly gang of summation signs.
And they are not as tough as they try to look like...

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

Instructions:
In the variance.py file create the following function as per the instructions given below:

```
def variance(x):
    """Computes the variance of a non-empty numpy.ndarray, using a for-loop.
    Args:
     x: has to be an numpy.ndarray, a vector.
    Returns:
     The variance as a float.
     None if x is an empty numpy.ndarray or if f is not a valid function.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21])
>>> variance(X)
134.57142857142858
>>> numpy.var(X)
134.57142857142858
>>>
>>> variance(X/2)
33.642857142857146
>>> numpy.var(X/2)
33.642857142857146
```

Exercise 03 - Standard Deviation

Turnin directory :	ex03
Files to turn in :	std.py
Forbidden function :	*.sum(), np.mean(), np.var(), np.std()
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - \frac{1}{m}\sum_{i=1}^{m}x_i)^2}$$

where:

- σ is the mathematical notation of the standard deviation, which is a real number
- x_i is the ith component of the vector x
- m is the number of components in x

As before, this formula is not as tough as it seems...

It can be reduced to something already known.

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2}$$

Seems familiar?

...

$$\sigma = \sqrt{\sigma^2}$$

Instructions:
In the variance.py file create the following function as per the instructions given below:

```
def std(x):
    """Computes the standard deviation of a non-empty numpy.ndarray, using a
for-loop.
    Args:
     x: has to be an numpy.ndarray, a vector.
    Returns:
     The standard deviation as a float.
     None if x is an empty numpy.ndarray or if f is not a valid function.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21])
>>> std(X)
11.600492600378166
>>> numpy.std(X)
11.600492600378166
>>>
>>> Y = np.array([2, 14, -13, 5, 12, 4, -19])
>>> std(Y)
11.410700312980492
>>> numpy.std(Y)
11.410700312980492
```

Exercise 04 - Dot product

Turnin directory :	ex04
Files to turn in :	dot.py
Forbidden function :	np.dot()
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$x \cdot y = x^T y = \sum_{i=1}^{m} x_i y_i$$

where:

- x is a vector of dimension m * 1
- y is a vector of dimension m * 1
- x_i is the ith component of the vector x
- y_i is the ith component of the vector y

Instructions:
In the dot.py file create the following function as per the instructions given below:

```
def dot(x, y):
    """Computes the dot product of two non-empty numpy.ndarray, using a
for-loop. The two arrays must have the same dimensions.
    Args:
     x: has to be an numpy.ndarray, a vector.
     y: has to be an numpy.ndarray, a vector.
    Returns:
     The dot product of the two vectors as a float.
     None if x or y are empty numpy.ndarray.
     None if x and y does not share the same dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21])
>>> Y = numpy.array([2, 14, -13, 5, 12, 4, -19])
>>> dot(X, Y)
917
>>> np.dot(X, Y)
917
>>>
>>> dot(X, X)
949
>>> np.dot(X, X)
949
>>>
>>> dot(Y, Y)
915
>>> np.dot(Y, Y)
915
```

Exercise 05 - Matrix-vector product

Turnin directory :	ex05
Files to turn in :	mat_vec_prod.py
Forbidden function :	np.dot()
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$(xy)_i = x_i \cdot y \text{ for i} = 0, 1, ..., \text{m}$$

where:

- x is a matrix of dimensions m * n
- y is a vector of dimension n * 1
- x_i is the ith component of the matrix x which is a row vector

Instructions:
In the mat_vec_prod.py file create the following function as per the instructions given below:

```
def mat_vec_prod(x, y):
    """Computes the product of two non-empty numpy.ndarray, using a
for-loop. The two arrays must have compatible dimensions.
    Args:
     x: has to be an numpy.ndarray, a matrix of dimension m * n.
     y: has to be an numpy.ndarray, a vector of dimension n * 1.
    Returns:
     The product of the matrix and the vector as a vector of dimension m *
1.
     None if x or y are empty numpy.ndarray.
     None if x and y does not share compatibles dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> W = numpy.array([
    [ -8,  8, -6, 14, 14, -9, -4],
    [ 2, -11, -2, -11, 14, -2, 14],
    [-13, -2, -5,  3, -8, -4, 13],
    [ 2, 13, -14, -15, -14, -15, 13],
    [ 2, -1, 12,  3, -7, -3, -6]])
>>> X = numpy.array([0, 15, -9, 7, 12, 3, -21]).reshape((7,1))
>>> Y = numpy.array([2, 14, -13, 5, 12, 4, -19]).reshape((7,1))
>>> mat_vec_prod(W, X)
array([[ 497],
       [-356],
       [-345],
       [-270],
       [ -69]])
>>> W.dot(X)
array([[ 497],
       [-356],
       [-345],
       [-270],
       [ -69]])
>>>
>>> mat_vec_mult(W, Y)
array([[ 452],
       [-285],
       [-333],
       [-182],
       [-133]])
>>> W.dot(Y)
array([[ 452],
       [-285],
       [-333],
       [-182],
       [-133]])
```

Exercise 06 - Matrix-matrix product

Turnin directory :	ex06
Files to turn in :	mat_mat_prod.py
Forbidden function :	np.dot()
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$(xy)^{(j)} = x \cdot y^{(j)} \text{ for j } = 0, 1, ..., \text{p}$$

where:

- x is a matrix of dimension m * n
- y is a matrix of dimension n * p
- j is the index of the columns
- $y^{(j)}$ is the jth column vector of the matrix y
- $xy^{(j)}$ is the jth column vector of the matrix xy

Instructions:
In the mat_mat_prod.py file create the following function as per the instructions given below:

```
def mat_mat_prod(x, y):
    """Computes the product of two non-empty numpy.ndarray, using a
for-loop. The two arrays must have compatible dimensions.
    Args:
     x: has to be an numpy.ndarray, a matrix of dimension m * n.
     y: has to be an numpy.ndarray, a vector of dimension n * p.
    Returns:
     The product of the matrices as a matrix of dimension m * p.
     None if x or y are empty numpy.ndarray.
     None if x and y does not share compatibles dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> W = numpy.array([
    [ -8,  8, -6, 14, 14, -9, -4],
    [ 2, -11, -2, -11, 14, -2, 14],
    [-13, -2, -5,  3, -8, -4, 13],
    [ 2, 13, -14, -15, -14, -15, 13],
    [ 2, -1, 12,  3, -7, -3, -6]])
>>> Z = array([
    [ -6, -1, -8, 7, -8],
        [ 7,  4,  0, -10, -10],
        [ 7, -13,  2,  2, -11],
        [ 3,  14,  7,  7, -4],
        [ -1, -3, -8, -4, -14],
        [ 9, -14,  9, 12, -7],
        [ -9, -4, -10, -3,  6]])
>>> mat_mat_prod(W, Z)
array([[ 45, 414, -3, -202, -163],
     [-294, -244, -367, -79,  62],
     [-107,  140,  13, -115,  385],
     [-302,  222, -302, -412,  447],
     [ 108, -33,  118,  79, -67]])
>>> W.dot(Z)
array([[ 45, 414, -3, -202, -163],
     [-294, -244, -367, -79,  62],
     [-107,  140,  13, -115,  385],
     [-302,  222, -302, -412,  447],
     [ 108, -33,  118,  79, -67]])
>>>
>>> mat_mat_prod(Z,W)
array([[ 148,  78, -116, -226, -76,    7,  45],
     [ -88, -108,  -30,  174,  364,  109, -42],
     [-126,  232, -186,  184,  -51,  -42, -92],
     [ -81,  -49, -227, -208,  112, -176,  390],
     [ 70,    3,  -60,  13,  162,  149, -110],
     [-207,  371, -323,  106, -261, -248,  83],
     [ 200, -53,  226, -49, -102,  156, -225]])
>>> Z.dot(W)
array([[ 148,  78, -116, -226, -76,    7,  45],
     [ -88, -108,  -30,  174,  364,  109, -42],
     [-126,  232, -186,  184,  -51,  -42, -92],
     [ -81,  -49, -227, -208,  112, -176,  390],
     [ 70,    3,  -60,  13,  162,  149, -110],
     [-207,  371, -323,  106, -261, -248,  83],
     [ 200, -53,  226, -49, -102,  156, -225]])
```

Exercise 07 - Mean Squared Error

Turnin directory :	ex07
Files to turn in :	mse.py
Forbidden function :	None
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

where:

- y is a vector of dimension m * 1,
- $\hat{y}$ is a vector of dimension m * 1,
- $\hat{y}_i$ is the ith component of $\hat{y}$,
- y_i is the ith component of vector y,

Instructions:
In the mse.py file create the following function as per the instructions given below:

```
def mse(y, y_hat):
    """Computes the mean squared error of two non-empty numpy.ndarray, using
a for-loop. The two arrays must have the same dimensions.
    Args:
     y: has to be an numpy.ndarray, a vector.
     y_hat: has to be an numpy.ndarray, a vector.
    Returns:
     The mean squared error of the two vectors as a float.
     None if y or y_hat are empty numpy.ndarray.
     None if y and y_hat does not share the same dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = np.array([0, 15, -9, 7, 12, 3, -21])
>>> Y = np.array([2, 14, -13, 5, 12, 4, -19])
>>> mse(X, Y)
4.285714285714286
>>> mse(X, X)
0.0
```

Exercise 08 - Vectorized Mean Squared Error

Turnin directory :	ex08
Files to turn in :	vec_mse.p
Forbidden function :	None
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$MSE = \frac{1}{m}(\hat{y} - y)^T(\hat{y} - y)$$

where:

- y is a vector of dimension m * 1,
- $\hat{y}$ is a vector of dimension m * 1,

Instructions:
In the vec_mse.py file create the following function as per the instructions given below:

```
def vec_mse(y, y_hat):
    """Computes the mean squared error of two non-empty numpy.ndarray,
without any for loop. The two arrays must have the same dimensions.
    Args:
     y: has to be an numpy.ndarray, a vector.
     y_hat: has to be an numpy.ndarray, a vector.
    Returns:
     The mean squared error of the two vectors as a float.
     None if y or y_hat are empty numpy.ndarray.
     None if y and y_hat does not share the same dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = np.array([0, 15, -9, 7, 12, 3, -21])
>>> Y = np.array([2, 14, -13, 5, 12, 4, -19])
>>> vec_mse(X, Y)
4.285714285714286
>>> vec_mse(X, X)
0.0
```

Exercise 09 - Mean Squared Error as linear cost function

Turnin directory :	ex09
Files to turn in :	linear_mse.py
Forbidden function :	
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

where:

- x is a matrix of dimension m * n,
- x_i is a vector of dimension n * 1,
- θ is a vector of dimension n * 1,
- $(h_\theta(x_i)$ is the dot product of the vectors θ and x_i,
- y is a vector of dimension m * 1,
- y_i is the ith component of y,

Instructions:
In the linear_mse.py file create the following function as per the instructions given below:

```
def linear_mse(y, x, theta):
    """Computes the mean squared error of three non-empty numpy.ndarray,
using a for-loop. The three arrays must have compatible dimensions.
    Args:
     y: has to be an numpy.ndarray, a vector of dimension m * 1.
     x: has to be an numpy.ndarray, a matrix of dimesion m * n.
     theta: has to be an numpy.ndarray, a vector of dimension n * 1.
    Returns:
     The mean squared error as a float.
     None if y, x, or theta are empty numpy.ndarray.
     None if y, x or theta does not share compatibles dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = np.array([
    [ -6, -7, -9],
        [ 13, -2, 14],
        [ -7, 14, -1],
        [ -8, -4, 6],
        [ -5, -9, 6],
        [ 1, -5, 11],
        [ 9, -11, 8]])
>>> Y = np.array([2, 14, -13, 5, 12, 4, -19])
>>> Z = np.array([3,0.5,-6])
>>> linear_mse(X, Y, Z)
2641.0
>>>
>>> W = np.array([0,0,0])
>>> linear_mse(X, Y, W)
130.71428571
```

Exercise 10 - Vectorized Mean Squared Error as a linear cost function

Turnin directory :	ex10
Files to turn in :	vec_linear_mse.py
Forbidden function :	None
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$MSE = \frac{1}{m}(X\theta - y)^T(X\theta - y)$$

where:

- x is a matrix of dimension m * n,
- θ is a vector of dimension n * 1,
- y is a vector of dimension m * 1,

Instructions:
In the vec_linear_mse.py file create the following function as per the instructions given below:

```
def vec_linear_mse(y, x, theta):
    """Computes the mean squared error of three non-empty numpy.ndarray,
without any for-loop. The three arrays must have compatible dimensions.
    Args:
     y: has to be an numpy.ndarray, a vector of dimension m * 1.
     x: has to be an numpy.ndarray, a matrix of dimesion m * n.
     theta: has to be an numpy.ndarray, a vector of dimension n * 1.
    Returns:
     The mean squared error as a float.
     None if y, x, or theta are empty numpy.ndarray.
     None if y, x or theta does not share compatibles dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = np.array([
    [ -6, -7, -9],
        [ 13, -2, 14],
        [ -7, 14, -1],
        [ -8, -4, 6],
        [ -5, -9, 6],
        [ 1, -5, 11],
        [ 9, -11, 8]])
>>> Y = np.array([2, 14, -13, 5, 12, 4, -19])
>>> Z = np.array([3,0.5,-6])
>>> vec_linear_mse(X, Y, Z)
2641.0
>>>
>>> W = np.array([0,0,0])
>>> vec_linear_mse(X, Y, W)
130.71428571
```

Exercise 11 - Linear Gradient - iterative version

Turnin directory :	ex11
Files to turn in :	gradient.py
Forbidden function :	None
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$\nabla(J)_j = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i^{(j)}$$

where

- $\nabla(J)$ is a vector of size n * 1
- x is a matrix of size m * n (i.e. a matrix containing m vectors of size n * 1)
- y is a vector of dimension m * 1
- θ is a vector of dimension n * 1
- x_i is the ith component of vector x
- y_i is the ith component of vector y
- $\nabla(J)_j$ is the jth component of $\nabla(J)$
- $h_\theta(x_i)$ is the result of the dot product of the vector θ and the vector x_i

Instructions:
In the gradient.py file create the following function as per the instructions given below:

```
def gradient(x, y, theta):
    """Computes a gradient vector from three non-empty numpy.ndarray, using
a for-loop. The two arrays must have the compatible dimensions.
    Args:
     x: has to be an numpy.ndarray, a matrice of dimension m * n.
     y: has to be an numpy.ndarray, a vector of dimension m * 1.
     theta: has to be an numpy.ndarray, a vector n * 1.
    Returns:
     The gradient as a numpy.ndarray, a vector of dimensions n * 1.
     None if x, y, or theta are empty numpy.ndarray.
     None if x, y and theta do not have compatible dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

Examples

```
>>> X = np.array([
    [ -6, -7, -9],
        [ 13, -2, 14],
        [ -7, 14, -1],
        [ -8, -4, 6],
        [ -5, -9, 6],
        [ 1, -5, 11],
        [ 9, -11, 8]])
>>> Y = np.array([2, 14, -13, 5, 12, 4, -19])
>>> Z = np.array([3,0.5,-6])
>>> gradient(X, Y, Z)
array([[ -18.67857143],
     [ 91.57142857],
     [-196.5     ]])
>>>
>>> W = np.array([0,0,0])
>>> gradient(X, Y, W)
array([[ 0.42857143],
     [ 11.64285714],
     [-13.21428571]])
>>>
>>> gradient(X, X.dot(Z), Z)
array([[0.],
     [0.],
     [0.]])
```

Exercise 12 - Linear Gradient - vectorized version

Turnin directory :	ex12
Files to turn in :	vec_gradient.py
Forbidden function :	None
Remarks :	n/a

Objective:
You must implement the following formula as a function:

$$\nabla(J) = \frac{1}{m} x^T (X\theta - y)$$

where

- $\nabla(J)$ is a vector of size n
- x is a matrix of size m * n
- y is a vector of size m * 1
- θ is a vector of size n * 1

Instructions:
In the vec_gradient.py file create the following function as per the instructions given below:

```
def vec_gradient(x, y, theta):
    """Computes a gradient vector from three non-empty numpy.ndarray,
without any for-loop. The three arrays must have the compatible dimensions.
    Args:
     x: has to be an numpy.ndarray, a matrice of dimension m * n.
     y: has to be an numpy.ndarray, a vector of dimension m * 1.
     theta: has to be an numpy.ndarray, a vector n * 1.
    Returns:
     The gradient as a numpy.ndarray, a vector of dimensions n * 1, containg
the result of the formula for all j.
     None if x, y, or theta are empty numpy.ndarray.
     None if x, y and theta do not have compatible dimensions.
    Raises:
     This function should not raise any Exception.
    """
```

`

Examples

```
>>> X = np.array([
    [ -6, -7, -9],
        [ 13, -2, 14],
        [ -7, 14, -1],
        [ -8, -4, 6],
        [ -5, -9, 6],
        [ 1, -5, 11],
        [ 9, -11, 8]])
>>> Y = np.array([2, 14, -13, 5, 12, 4, -19])
>>> Z = np.array([3,0.5,-6])
>>> vec_gradient(X, Y, Z)
array([[ -18.67857143],
     [ 91.57142857],
     [-196.5     ]])
>>>
>>> W = np.array([0,0,0])
>>> vec_gradient(X, Y, W)
array([[ 0.42857143],
     [ 11.64285714],
     [-13.21428571]])
>>>
>>> vec_gradient(X, X.dot(Z), Z)
array([[0.],
     [0.],
     [0.]])
```