bioeng-ml-python

2019년 12월 13일

1 파이썬 기본 문법

- 파이썬의 변수는 값을 저장하는 주소를 가리키는 바인더

```
In [15]: a = 10
         print(a)
         print(type(a))
```

```
10
<class 'int'>
```

- 논리연산자, True, False, and, or

```
In [2]: print(True and False)
        print(True or False)
        print(not True and False)
```

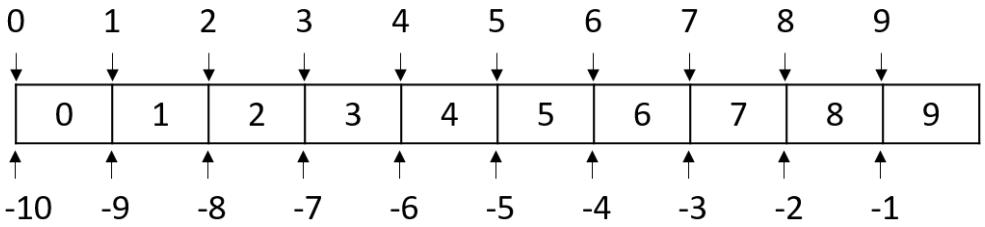
```
False
True
False
```

- 조건문, if, elif, else

```
In [3]: a = 10
        if a < 0:
            print("Negative")
        elif a >= 0 and a < 10:
            print("Less than 10")
        else:
            print("Geater than 10")
            print("Or equal to 10")
```

```
Geater than 10
Or equal to 10
```

- 반복문 for, while



indexing.png

```
In [4]: for i in [0, 1, 2, 3]:
                print("for1", i)
                print("for2", i)

        a = 4
        i = 0
        while i < a:
            print("while", i)
            if i == 2:
                print("stop")
                break
            i = i+1
```

```
for1 0
for2 0
for1 1
for2 1
for1 2
for2 2
for1 3
for2 3
while 0
while 1
while 2
stop
```

2 파이썬 기본 자료 구조

2.1 List (리스트)

- 리스트는 여러 개의 데이터를 순서대로 저장하고 관리할 때 사용

```
In [16]: expression = ["geneA", 1]
         expression = [1, 2, 3]
         expression = []
```

- 인덱싱은 값 자체 (1은 두 번째값)
- 슬라이싱은 값 사이 경계선 (1은 첫 번째 값과 두 번째 값 사이)

```
In [6]: geneids = [x for x in range(10)] # 리스트 컴프리헨션
        print(geneids[0])
```

```
        print(geneids[-1])
        print(geneids[0:2])
        print(geneids[:])
        print(geneids[:-1])
        print(geneids[1:])
        print(geneids[:-10])
```

```
0
9
[0, 1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[]
```

- 리스트 데이터 삽입 삭제

```
In [12]: geneids = [1, 2, 3]
         print(geneids)
         geneids.append(4)
         print(geneids)
         print("length: %d" % len(geneids))
         geneids[len(geneids):] = [5]
         print(geneids)
         print(geneids.pop())
         print(geneids)
```

```
[1, 2, 3]
[1, 2, 3, 4]
length: 4
[1, 2, 3, 4, 5]
5
[1, 2, 3, 4]
```

2.2 Tuple (튜플)

- 리스트와 같은 기능이지만 '(', ')'를 사용하고 원소를 변경할 수 없음
- 리스트보다 빠른 속도, 리스트와 동일한 인덱싱 방법

```
In [17]: geneids = (1, 2, 3)
         print(geneids[0:2])
         #geneids[0] = 4 ## error
```

```
(1, 2)
```

- 반복문에서 리스트 또는 튜플 활용

```
In [18]: geneids = ['123', '456', '789']
         for geneid in geneids:
             print("geneid: %s" %geneid)
```

```
geneid: 123
geneid: 456
geneid: 789
```

2.3 Dictionary (딕셔너리)

- 키(key)와 값(value)을 쌍으로 저장, '{'와 '}'를 사용

```
In [19]: gene_expr = {}
         gene_expr['A'] = 0.5
         print(gene_expr)
         gene_expr['B'] = 1.2
         print(gene_expr)
         print(len(gene_expr))
```

```
{'A': 0.5}
{'A': 0.5, 'B': 1.2}
2
```

- 인덱싱은 '[', ']' 사용, 키 값으로 인덱싱, 정수값 인덱싱 불가

```
In [17]: print(gene_expr['A'])
         ## gene_expr[0] # error
```

```
0.5
```

- 데이터 추가는 key값 value값으로 수행, 삭제는 del 함수 이용

```
In [18]: gene_expr['C'] = 0.3
         print(gene_expr)
         del gene_expr['C']
         print(gene_expr)
```

```
{'A': 0.5, 'B': 1.2, 'C': 0.3}
{'A': 0.5, 'B': 1.2}
```

- key 값과 value 값 구하기

```
In [20]: gene_expr_keys = list(gene_expr.keys())
         print("keys:", gene_expr_keys)
         gene_expr_values = list(gene_expr.values())
         print("values:", gene_expr_values)
```

```
keys: ['A', 'B']
values: [0.5, 1.2]
```

- in 활용 키 값 탐색

```
In [26]: print('D' in gene_expr_keys)
         print('D' in gene_expr)
         print('A' in gene_expr)
```

```
False
False
True
```

- 반복문에서 딕셔너리 활용 items()

```
In [2]: gene_expr = {'A':0.5, 'B':1.2, 'C':0.3, 'D':3.2}
        for geneid, expval in gene_expr.items():
            print("%s expression value is %s" %(geneid, expval))
```

```
A expression value is 0.5
B expression value is 1.2
C expression value is 0.3
D expression value is 3.2
```

3 파이썬 함수, 모듈, 클래스

3.1 함수

- 리스트 값 평균 리턴하는 함수

```
In [20]: def average(input):
             if len(input) == 0:
                 return None
             return sum(input) / len(input)

         x = [1,2,3,4,5,6,7,8,9,10]
         print(average(x))
```

```
5.5
```

3.2 모듈

- 위 average 함수를 mystat.py 라는 이름의 파일로 저장, 모듈로 활용

```
In [28]: import mystat
         x = list(range(10))
         print(mystat.average(x))
```

```
4.5
```

- 모듈 직접 실행시 모듈 내 test 코드 실행 (name == main, True)

```
In [21]: %run mystat
```

```
average function is working well
```

- 모듈 임포트

```
In [22]: import os
         os.getcwd()
```

```
Out[22]: '/home/bioengml'
```

```
In [23]: from os import getcwd
         getcwd()
```

```
Out[23]: '/home/bioengml'
```

3.3 클래스

- Gene, Strain class 생성 연습
- Gene attribute: name, chromosomal location, length
- Strain attribute (변수): name, length of chromosome
- Strain method (함수): compute average length of the genes

```
In [24]: import statistics
         class ORF:
             def __init__(self, location, length, seq):
                 self.location = location
                 self.length = length
                 self.sequence = seq

         class Strain:
             def __init__(self, name, chrlength):
                 self.name = name
                 self.chr_length = chrlength
                 self.orfs = []
             def add_orf(self, location, length, seq):
                 self.orfs.append(ORF(location, length, seq))
             def gene_average(self):
                 return statistics.mean([s.length for s in self.orfs])
```

```
In [25]: ecoli = Strain("ecoli", 5000000)
         ecoli.add_orf(1, 1000, "ATG")
         ecoli.add_orf(1001, 2000, "CCT")
         ecoli.add_orf(3001, 3000, "ATC")
```

```
In [26]: print([g.location for g in ecoli.orfs])
         print([g.sequence for g in ecoli.orfs])
```

```
[1, 1001, 3001]
['ATG', 'CCT', 'ATC']
```

- 상속

```
In [27]: class Gene(ORF):
             def add_protein(self, prot_name, prot_seq):
                 self.prot_name = prot_name
                 self.prot_sequence = prot_seq
```

```
In [28]: gene1 = Gene(1, 1000, "ATG")
         print(gene1.location)
         gene1.add_protein("myprotein", "M")
         print(gene1.prot_name)
```

```
1
myprotein
```

4 파일 읽기 쓰기

```
In [30]: f = open("README.md", 'rt')
         lines = f.readlines()
         for line in lines:
             nline = line.split('\n')[0]
             print(nline)
```

```
# 한국생물공학회 교육워크샵
12월 한국생물공학회 교육워크샵의 [생물공학 기계학습]  실습 관련 사이트 입니다.
```

```
In [31]: f = open("write_test.txt", 'wt')
         f.write('gene1;')
         f.write('1;')
         f.write('1000')
         f.close()
```

```
In [32]: f = open("write_test.txt", 'rt')
         lines = f.readlines()
         for line in lines:
             nline = line.split(';')
             print(nline)
```

```
['gene1', '1', '1000']
```

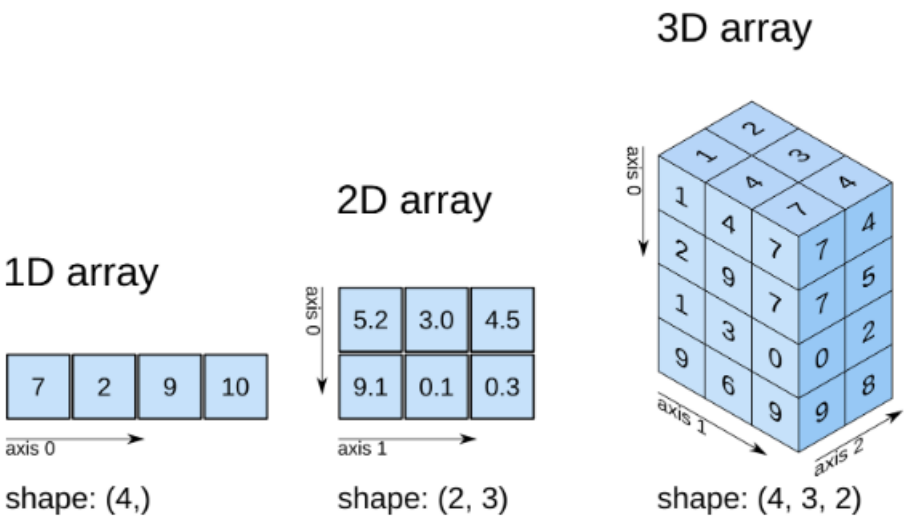
5 Numpy 자료구조 ndarray

- 행렬이나 다차원 배열 처리용 파이썬 라이브러리
- 같은 타입의 데이터만 허용
- 리스트에 비해 20배 이상 빠른 속도

```
In [1]: import numpy as np
```

```
In [2]: arr = [1, 2, 3]
        print(arr)
        print(type(arr))

        a = np.array([1,2,3])
        print(a)
        print(a.dtype)
        print(a.shape)
        print(type(a))
```

```
[1, 2, 3]
<class 'list'>
[1 2 3]
int64
(3,)
```



from https://www.oreilly.com/library/view/elegant-scipy/9781491922927/ch01.html

```
<class 'numpy.ndarray'>
```

- numpy 자료형
- 부호가 있는 정수 int(8, 16, 32, 64)
- 부호가 없는 정수 uint(8 ,16, 32, 54)
- 실수 float(16, 32, 64, 128)
- 복소수 complex(64, 128, 256)
- 불리언 bool
- 문자열 string_
- 파이썬 오프젝트 object
- 유니코드 unicode_

- np.zeros(), np.ones(), np.arange()
- 행렬 연산 지원

```
In [35]: a = np.arange(1, 10).reshape(3,3)
         print(a)
         a = np.ones((3,4), dtype=np.int16)
         b = np.ones((3,4), dtype=np.int16)
         print(a)
         print(b)
         print(a+b)
         print(a-b)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[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]]
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

- numpy 함수
- np.sqrt()
- np.log()
- np.square()
- np.log()
- np.ceil()
- np.floor()
- np.isnan()
- np.sum()
- np.mean()
- np.std()
- np.min()

6 Pandas 자료구조 Series, DataFrame

- Pandas의 Series는 1차원, DataFrame은 2차원 데이터를 다루는 자료구조
- 리스트와 딕셔너리의 조합형
- 숫자형, 문자형, 범주형 등의 다양한 데이터 입력 가능

```
In [36]: from pandas import Series, DataFrame
```

```
In [37]: genes = Series([0.1, 0.2, 1.4, 0.6, 1.1])
         print(genes)
```

```
0    0.1
1    0.2
2    1.4
3    0.6
4    1.1
dtype: float64
```

```
In [38]: genes = Series([0.1, 0.2, 1.4, 0.6, 1.1], index=['A', 'B', 'C', 'D', 'E'])
         print(genes)
```

```
A    0.1
B    0.2
C    1.4
D    0.6
E    1.1
dtype: float64
```

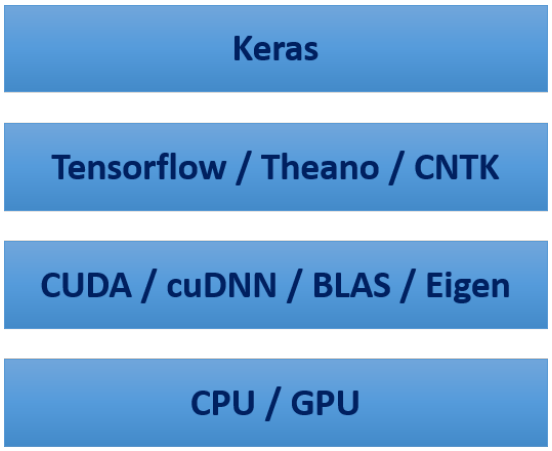
- 인덱스 자동 정렬, 행렬 연산

```
In [39]: genes1 = Series([0.1, 0.2, 1.4, 0.6, 1.1], index=['A', 'B', 'C', 'D', 'E'])
         genes2 = Series([0.1, 0.2, 1.4, 0.6, 1.1], index=['B', 'C', 'D', 'E', 'A'])
         genes1 + genes2
```

```
Out[39]: A    1.2
         B    0.3
         C    1.6
         D    2.0
         E    1.7
         dtype: float64

In [40]: print(genes2.sort_values())
         print(genes2.sort_index())

B    0.1
C    0.2
E    0.6
A    1.1
D    1.4
dtype: float64
A    1.1
B    0.1
C    0.2
D    1.4
E    0.6
dtype: float64
```

- DataFrame 생성은 '{', '}' 이용
- DataFrame은 Series의 집합

```
In [41]: genes = {'A': [0.5, 0.1, 0.3],
                  'B': [0.8, 0.9, 0.4]}
         print(genes)
         genes_df = DataFrame(genes)
         print(genes_df)
         print(genes_df['A'])
         print(type(genes_df['A']))

{'A': [0.5, 0.1, 0.3], 'B': [0.8, 0.9, 0.4]}
     A    B
0  0.5  0.8
1  0.1  0.9
2  0.3  0.4
0    0.5
1    0.1
2    0.3
Name: A, dtype: float64
<class 'pandas.core.series.Series'>


In [42]: genes = {'A': [0.5, 0.1, 0.3],
                  'B': [0.8, 0.9, 0.4]}
         genes_df = DataFrame(genes, columns=['B', 'A'], index=['day1', 'day2', 'day3'])
         print(genes_df)

        B    A
day1  0.8  0.5
day2  0.9  0.1
```




```
day3  0.4  0.3
```

```
In [43]: print(genes_df['A'])
         print(genes_df.loc['day1'])
         print(genes_df.index)
         print(list(genes_df.columns))
```

```
day1    0.5
day2    0.1
day3    0.3
Name: A, dtype: float64
B    0.8
A    0.5
Name: day1, dtype: float64
Index(['day1', 'day2', 'day3'], dtype='object')
['B', 'A']
```

7 다차원 numpy 자료구조 텐서 (Tensor)

- 딥러닝 프레임워크 (라이브러리 모듈 묶어놓은 패키지)
- tensorflow - 구글 개발, 가장 높은 인기
- theano - python 기반 최초 딥러닝 라이브러리
- PyTorch - 페이스북 개발, 낮은 진입 장벽
- CNTK (Cognitive Toolkit) - Microsoft 개발, 높음 성능
- 참고로 Keras는 tensorflow, theano, CNTK를 백엔드엔진으로 사용해서 동작하는 고수준 라이브러리

- 텐서는 수치형 (float32, uint8, float64) 데이터를 주로 다룸
- 임의의 차원 개수를 가지는 행렬의 일반화된 모습
- 0D 텐서는 스칼라, 1D 텐서는 벡터, 2D 텐서는 행렬, ...
- 랭크(ndim), 크기(shape), 타입(dtype) 속성이 있음

```
In [44]: import numpy as np
         x = np.array(12)
         print(x)
         print(x.ndim)
```

```
12
0
```

```
In [45]: x = np.array([1, 2, 3, 4, 5])
         print(x.ndim)
```

```
1
```

```
In [46]: x = np.array([[1,2,3,4,5],
                       [6,7,8,9,10],
                       [11,12,13,14,15]])
         print(x.ndim)
```

```
2
```

```
In [47]: x = np.array([[[1,2,3],
                        [2,3,4]],
                       [[5,6,7],
                        [8,9,10]],
                       [[11,12,13],
                        [14,15,16]]])
```

```
In [48]: x.shape
```

```
Out[48]: (3, 2, 3)
```

```
In [49]: from keras.datasets import mnist
         (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Using TensorFlow backend.
```

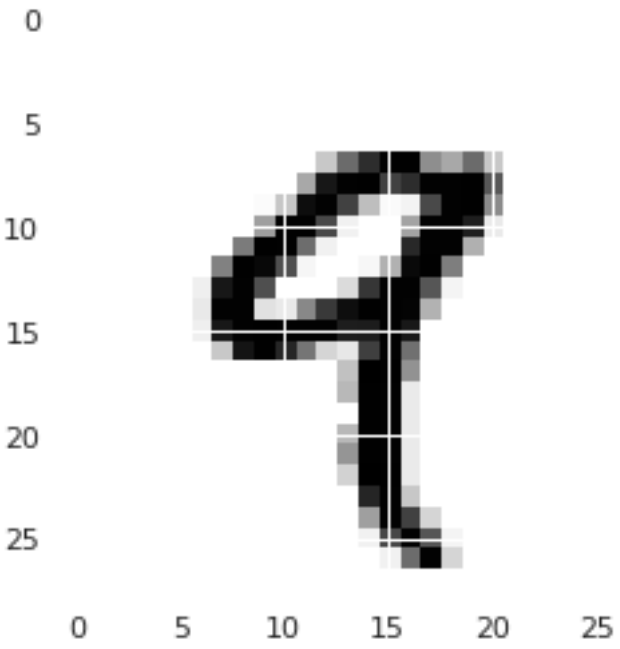
```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 8s 1us/step
```

```
In [50]: print(train_images.shape)
         print(train_labels.shape)
         print(train_images.dtype)
```

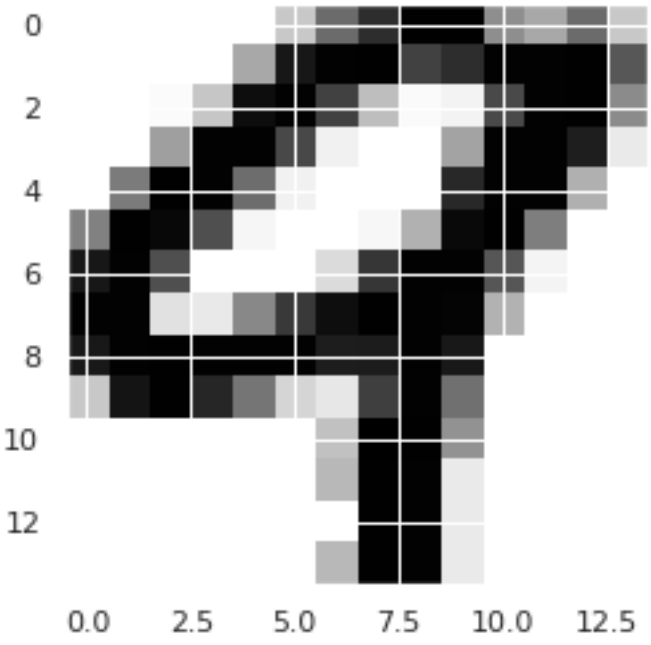
```
(60000, 28, 28)
(60000,)
uint8
```

```
In [51]: digit = train_images[4]
         print(digit.shape)

         import matplotlib.pyplot as plt
         plt.imshow(digit, cmap=plt.cm.binary)
         plt.show()
```

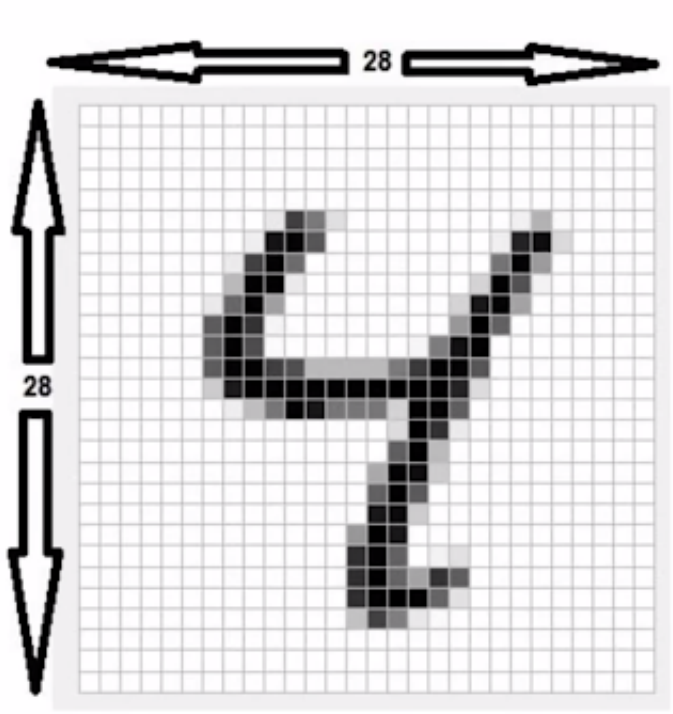
```
(28, 28)
```




```
In [52]: my_slice = train_images[4,7:-7,7:-7]
         plt.imshow(my_slice, cmap=plt.cm.binary)
         plt.show()
```



- 배치 (batch) - 훈련 데이터를 나누어 입력



"y_mnist"

- 에폭 (epoch) - 전체 훈련 데이터에 수행되는 각 반복

```
In [53]: batch1 = train_images[:128]
         batch2 = train_images[129:256]
         #...
```

8 Neural Network 예제

- 유명 예제 중 하나인 MNIST 예제
- 흑백 손글씨 숫자 이미지 (28x28픽셀)을 10개 범주에서 (0 부터 9까지)로 분류하는 문제
- 미국 국립표준기술연구소 (NIST)에서 수집한 60000개 훈련 이미지와 1만개 테스트 이미지 구성

```
In [11]: from keras.datasets import mnist
         (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

In [12]: print(train_images.shape)
         print(train_labels.shape)
         print(test_images.shape)
         print(test_labels.shape)

(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

- 배열 변환

```
In [13]: train_images = train_images.reshape((60000, 28*28))
         train_images = train_images.astype('float32')/255
         test_images = test_images.reshape((10000, 28*28))
         test_images = test_images.astype('float32')/255

In [14]: print(train_images.shape)
         print(train_labels.shape)
         print(test_images.shape)
         print(test_labels.shape)

(60000, 784)
(60000,)
(10000, 784)
(10000,)


In [15]: print(test_labels[:10])

[7 2 1 0 4 1 4 9 5 9]
```

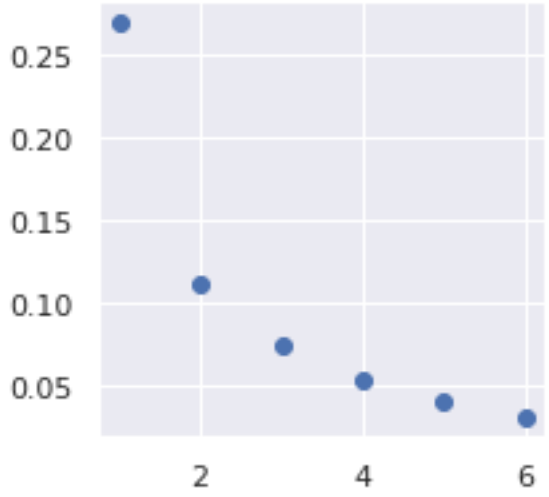
- 신경망

```
In [16]: from keras import models
         from keras import layers
         from keras.utils import to_categorical

         network = models.Sequential()
         network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
         network.add(layers.Dense(10, activation='softmax'))

         network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
         train_labels = to_categorical(train_labels)
         test_labels = to_categorical(test_labels)

In [17]: print(test_labels.shape)
         print(test_labels[:10,])

(10000, 10)
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]


In [18]: history = network.fit(train_images, train_labels, epochs=6, batch_size=128)

Epoch 1/6
60000/60000 [==============================] - 8s 131us/step - loss: 0.2694 - acc: 0.9215
Epoch 2/6
```

```
60000/60000 [==============================] - 7s 122us/step - loss: 0.1115 - acc: 0.9668
Epoch 3/6
60000/60000 [==============================] - 6s 107us/step - loss: 0.0741 - acc: 0.9776
Epoch 4/6
60000/60000 [==============================] - 7s 109us/step - loss: 0.0534 - acc: 0.9841
Epoch 5/6
60000/60000 [==============================] - 8s 136us/step - loss: 0.0410 - acc: 0.9880
Epoch 6/6
60000/60000 [==============================] - 8s 135us/step - loss: 0.0312 - acc: 0.9902
```

```
In [93]: import matplotlib.pyplot as plt

         history_dict = history.history
         print(history_dict.keys())
         loss =history_dict['loss']
         acc = history_dict['acc']
         epochs = range(1, len(loss)+1)
         plt.rcParams["figure.figsize"] = (3,3)
         plt.plot(epochs, loss, 'bo')
         plt.show()
```

```
dict_keys(['loss', 'acc'])
```



- 테스트 세트에서 모델 작동 확인

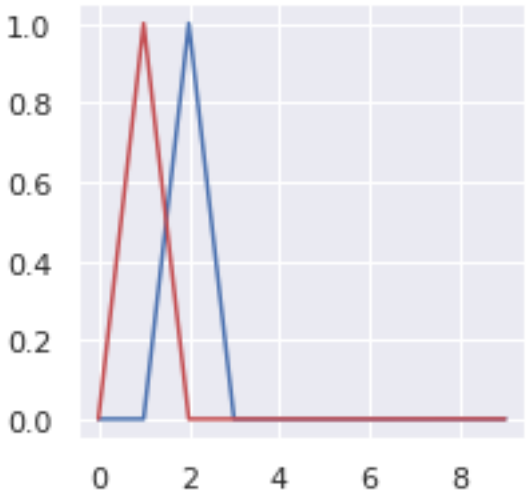
```
In [87]: results = network.predict(test_images)
         print(results.shape)
```

```
(10000, 10)
```

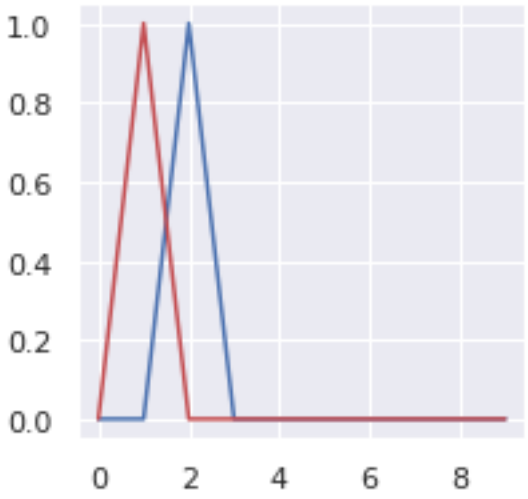
```
In [90]: import matplotlib.pyplot as plt
         plt.rcParams["figure.figsize"] = (3,3)
```

```
        print([round(x) for x in list(results[1,])])
        plt.plot(results[1,], "b")
        plt.plot(results[2,], "r")
        plt.show()
```

```
[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```



In [91]:
```
        plt.plot(test_labels[1,], "b")
        plt.plot(test_labels[2,], "r")
        plt.show()
        print(test_labels[1,])
```




```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

```
In [23]: results = network.evaluate(test_images, test_labels)
         print(results)

10000/10000 [==============================] - 1s 84us/step
[0.062462628302245866, 0.9827]
```

9 Biopython - Sequence objects

```
In [24]: from Bio.Seq import Seq
         from Bio.Alphabet import IUPAC
         my_seq = Seq("AGTACACTGGT", IUPAC.unambiguous_dna)
         my_seq

Out[24]: Seq('AGTACACTGGT', IUPACUnambiguousDNA())

In [25]: for index, letter in enumerate(my_seq):
             print("%i %s " % (index, letter))

0 A
1 G
2 T
3 A
4 C
5 A
6 C
7 T
8 G
9 G
10 T


In [26]: x = [1, 4, 5, 7, 8]
         for i in enumerate(x):
             print(i)

(0, 1)
(1, 4)
(2, 5)
(3, 7)
(4, 8)


In [27]: print(my_seq)
         print(my_seq[0:3])
         print(my_seq[0::2])
         print(str(my_seq))
         print(my_seq + "ATG")
         print(my_seq=="ATG")
         print("AGT" in my_seq)
         my_seq_low = my_seq.lower()
         print(my_seq_low)
         print(my_seq_low.upper())
         print(my_seq.complement())
         print(my_seq.reverse_complement())
```

```
AGTACACTGGT
AGT
ATCCGT
AGTACACTGGT
AGTACACTGGTATG
False
True
agtacactggt
AGTACACTGGT
TCATGTGACCA
ACCAGTGTACT
```

```
In [29]: mrna = my_seq.transcribe()
         print(mrna)
         prot = mrna.translate() ## truncated
         print(prot)
         print(my_seq.translate())
```

```
AGUACACUGGU
STL
STL
```

```
In [30]: from Bio.Data import CodonTable
         standard_table = CodonTable.unambiguous_dna_by_id[1]
         print(standard_table)
         print(standard_table.start_codons)
         print(standard_table.stop_codons)
```

```
Table 1 Standard, SGC0

  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA Stop| A
T | TTG L(s)| TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L(s)| CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I   | ACT T   | AAT N   | AGT S   | T
A | ATC I   | ACC T   | AAC N   | AGC S   | C
A | ATA I   | ACA T   | AAA K   | AGA R   | A
A | ATG M(s)| ACG T   | AAG K   | AGG R   | G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V   | GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
['TTG', 'CTG', 'ATG']
```

```
['TAA', 'TAG', 'TGA']
```

10 biopython - SeqRecord

- Sequence annotation objects
- 특정 서열의 identifier나 feature 정보 포함

```
In [31]: from Bio.Seq import Seq
         from Bio.SeqRecord import SeqRecord

         simple_seq = Seq("GATC")
         simple_seq_r = SeqRecord(simple_seq)

In [32]: simple_seq_r.id = "AC12345"
         simple_seq_r.description = "Made up sequence I wish I could write a paper about"
         print(simple_seq_r.description)
         print(simple_seq_r.seq)

Made up sequence I wish I could write a paper about
GATC


In [ ]: help(SeqRecord)
```

- Read fasta file
- *Yersinia pestis biovar Microtus* str. 91001 plasmid (페스트균)

```
In [34]: from Bio import SeqIO
         record = SeqIO.read("datasets/NC_005816.fna", "fasta")

In [35]: record

Out[35]: SeqRecord(seq=Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', SingleLetter

In [36]: print(record.id)
         print(record.name)
         print(record.description)

gi|45478711|ref|NC_005816.1|
gi|45478711|ref|NC_005816.1|
gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequenc
```

- Read GenBank file

```
In [37]: from Bio import SeqIO
         record = SeqIO.read("datasets/NC_005816.gb", "genbank")

In [38]: print(record.id)
         print(record.name)
         print(record.description)
         print(len(record.features))
         print(record.features[0])
         print(record.features[2])
         print(record.features[3])
```

```
NC_005816.1
NC_005816
Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence
41
type: source
location: [0:9609](+)
qualifiers:
    Key: biovar, Value: ['Microtus']
    Key: db_xref, Value: ['taxon:229193']
    Key: mol_type, Value: ['genomic DNA']
    Key: organism, Value: ['Yersinia pestis biovar Microtus str. 91001']
    Key: plasmid, Value: ['pPCP1']
    Key: strain, Value: ['91001']

type: gene
location: [86:1109](+)
qualifiers:
    Key: db_xref, Value: ['GeneID:2767718']
    Key: locus_tag, Value: ['YP_pPCP01']

type: CDS
location: [86:1109](+)
qualifiers:
    Key: codon_start, Value: ['1']
    Key: db_xref, Value: ['GI:45478712', 'GeneID:2767718']
    Key: locus_tag, Value: ['YP_pPCP01']
    Key: note, Value: ['similar to corresponding CDS from previously sequenced pPCP plasmid of Yersinia
    Key: product, Value: ['putative transposase']
    Key: protein_id, Value: ['NP_995567.1']
    Key: transl_table, Value: ['11']
    Key: translation, Value: ['MVTFETVMEIKILHKQGMSSRAIARELGISRNTVKRYLQAKSEPPKYTPRPAVASLLDEYRDYIRQRIADAH
```

- 위치 탐색 in 활용

```
In [39]: for feature in record.features:
             if 4350 in feature:
                 print(feature)
                 print("%s %s" % (feature.type, feature.qualifiers.get("db_xref")))
```

```
type: source
location: [0:9609](+)
qualifiers:
    Key: biovar, Value: ['Microtus']
    Key: db_xref, Value: ['taxon:229193']
    Key: mol_type, Value: ['genomic DNA']
    Key: organism, Value: ['Yersinia pestis biovar Microtus str. 91001']
    Key: plasmid, Value: ['pPCP1']
    Key: strain, Value: ['91001']

source ['taxon:229193']
type: gene
location: [4342:4780](+)
qualifiers:
```

```
    Key: db_xref, Value: ['GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']

gene ['GeneID:2767712']
type: CDS
location: [4342:4780](+)
qualifiers:
    Key: codon_start, Value: ['1']
    Key: db_xref, Value: ['GI:45478716', 'GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']
    Key: note, Value: ['similar to many previously sequenced pesticin immunity protein entries of Yersi
    Key: product, Value: ['pesticin immunity protein']
    Key: protein_id, Value: ['NP_995571.1']
    Key: transl_table, Value: ['11']
    Key: translation, Value: ['MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNSLSHGIYGKQTTFKQTEFTNIKSNTKKH

CDS ['GI:45478716', 'GeneID:2767712']
```

- format 함수

```
In [ ]: record.format("fasta")
```

- Slicing

```
In [41]: print(len(record))
         print(len(record.features))

9609
41
```

```
In [42]: print(record.features[20])
         print(record.features[21])

type: gene
location: [4342:4780](+)
qualifiers:
    Key: db_xref, Value: ['GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']

type: CDS
location: [4342:4780](+)
qualifiers:
    Key: codon_start, Value: ['1']
    Key: db_xref, Value: ['GI:45478716', 'GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']
    Key: note, Value: ['similar to many previously sequenced pesticin immunity protein entries of Yersi
    Key: product, Value: ['pesticin immunity protein']
    Key: protein_id, Value: ['NP_995571.1']
    Key: transl_table, Value: ['11']
```

```
    Key: translation, Value: ['MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNSLSHGIYGKQTTFKQTEFTNIKSNTKKH
```

```
In [43]: sub_record = record[4300:4800]
         print(sub_record)
         print(len(sub_record))
         print(len(sub_record.features))
         print(sub_record.features[0])
         print(sub_record.features[1])
```

```
ID: NC_005816.1
Name: NC_005816
Description: Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence
Number of features: 2
Seq('ATAAATAGATTATTCCAAATAATTTATTTATGTAAGAACAGGATGGGAGGGGGA...TTA', IUPACAmbiguousDNA())
500
2
type: gene
location: [42:480](+)
qualifiers:
    Key: db_xref, Value: ['GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']

type: CDS
location: [42:480](+)
qualifiers:
    Key: codon_start, Value: ['1']
    Key: db_xref, Value: ['GI:45478716', 'GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']
    Key: note, Value: ['similar to many previously sequenced pesticin immunity protein entries of Yersi
    Key: product, Value: ['pesticin immunity protein']
    Key: protein_id, Value: ['NP_995571.1']
    Key: transl_table, Value: ['11']
    Key: translation, Value: ['MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNSLSHGIYGKQTTFKQTEFTNIKSNTKKH
```

11 Biopython - Parsing Genbank records from the NCBI

- 파일 읽기/쓰기 with 문 사용

```
In [44]: from Bio import Entrez
         from Bio import SeqIO
         Entrez.email = "haseong@kribb.re.kr"
         with Entrez.efetch(db="nucleotide", rettype="gb", retmode="text", id="6273291") as handle:
             seq_record = SeqIO.read(handle, "gb")
         print("%s with %i features" % (seq_record.id, len(seq_record.features)))
```

```
AF191665.1 with 3 features
```

- 여러개 record에 대해서는 parse 함수를 사용

```
In [45]: with Entrez.efetch(db="nucleotide", rettype="gb", retmode="text", id="6273291,6273290,6273289"
             for seq_record in SeqIO.parse(handle, "gb"):
                 print("%s %s..." % (seq_record.id, seq_record.description[:50]))
                 print("Sequence length %i, %i features, from: %s"
                       % (len(seq_record), len(seq_record.features), seq_record.annotations["source"]))
```

```
AF191665.1 Opuntia marenae rpl16 gene; chloroplast gene for c...
Sequence length 902, 3 features, from: chloroplast Grusonia marenae
AF191664.1 Opuntia clavata rpl16 gene; chloroplast gene for c...
Sequence length 899, 3 features, from: chloroplast Grusonia clavata
AF191663.1 Opuntia bradtiana rpl16 gene; chloroplast gene for...
Sequence length 899, 3 features, from: chloroplast Grusonia bradtiana
```

- 파일 쓰기

```
In [47]: from Bio import SeqIO
         SeqIO.write(seq_record, "datasets/my_seq_records.fa", "fasta")
```

```
Out[47]: 1
```

12 Biopython - multiple sequence alignment objects

- http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc70
- Bio.AlignIO.read() returns a single MultipleSeqAlignment object
- Bio.AlignIO.parse() returns MultipleSeqAlignment objects
- Alignment tools

```
In [48]: import Bio.Align.Applications as alnapps
         dir(alnapps)
```

```
Out[48]: ['ClustalOmegaCommandline',
          'ClustalwCommandline',
          'DialignCommandline',
          'MSAProbsCommandline',
          'MafftCommandline',
          'MuscleCommandline',
          'PrankCommandline',
          'ProbconsCommandline',
          'TCoffeeCommandline',
          '_ClustalOmega',
          '_Clustalw',
          '_Dialign',
          '_MSAProbs',
          '_Mafft',
          '_Muscle',
          '_Prank',
          '_Probcons',
          '_TCoffee',
          '__all__',
          '__builtins__',
          '__cached__',
```

```
          '__doc__',
          '__file__',
          '__loader__',
          '__name__',
          '__package__',
          '__path__',
          '__spec__']
```

- Clustalw를 이용한 서열 정렬 (cactus family Opuntia(선인장))

```
In [49]: from Bio.Align.Applications import ClustalwCommandline as clw
         #help(clw)
         cline = clw("clustalw2", infile="datasets/opuntia.fasta")
         stdout, stderr = cline()
         print(cline)
         print(stdout)
```

```
clustalw2 -infile=datasets/opuntia.fasta



 CLUSTAL 2.1 Multiple Sequence Alignments


Sequence format is Pearson
Sequence 1: gi|6273291|gb|AF191665.1|AF191665   902 bp
Sequence 2: gi|6273290|gb|AF191664.1|AF191664   899 bp
Sequence 3: gi|6273289|gb|AF191663.1|AF191663   899 bp
Sequence 4: gi|6273287|gb|AF191661.1|AF191661   895 bp
Sequence 5: gi|6273286|gb|AF191660.1|AF191660   893 bp
Sequence 6: gi|6273285|gb|AF191659.1|AF191659   894 bp
Sequence 7: gi|6273284|gb|AF191658.1|AF191658   896 bp
Start of Pairwise alignments
Aligning...

Sequences (1:2) Aligned. Score:  99
Sequences (1:3) Aligned. Score:  99
Sequences (1:4) Aligned. Score:  98
Sequences (1:5) Aligned. Score:  98
Sequences (1:6) Aligned. Score:  98
Sequences (1:7) Aligned. Score:  98
Sequences (2:3) Aligned. Score:  99
Sequences (2:4) Aligned. Score:  98
Sequences (2:5) Aligned. Score:  98
Sequences (2:6) Aligned. Score:  98
Sequences (2:7) Aligned. Score:  98
Sequences (3:4) Aligned. Score:  98
Sequences (3:5) Aligned. Score:  98
Sequences (3:6) Aligned. Score:  98
Sequences (3:7) Aligned. Score:  98
Sequences (4:5) Aligned. Score:  99
Sequences (4:6) Aligned. Score:  99
Sequences (4:7) Aligned. Score:  99
Sequences (5:6) Aligned. Score:  99
Sequences (5:7) Aligned. Score:  99
```

```
Sequences (6:7) Aligned. Score:  99
Guide tree file created:   [datasets/opuntia.dnd]

There are 6 groups
Start of Multiple Alignment

Aligning...
Group 1: Sequences:   2      Score:16933
Group 2: Sequences:   2      Score:16703
Group 3: Sequences:   4      Score:16812
Group 4: Sequences:   2      Score:17071
Group 5: Sequences:   3      Score:16845
Group 6: Sequences:   7      Score:16678
Alignment Score 114256

CLUSTAL-Alignment file created  [datasets/opuntia.aln]
```

```
In [50]: #print(stdout)
         from Bio import AlignIO
         align = AlignIO.read("datasets/opuntia.aln", "clustal")
         print(align)
```

```
SingleLetterAlphabet() alignment with 7 rows and 906 columns
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273285|gb|AF191659.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273284|gb|AF191658.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273287|gb|AF191661.1|AF191
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273286|gb|AF191660.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273290|gb|AF191664.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273289|gb|AF191663.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273291|gb|AF191665.1|AF191
```

```
In [52]: from Bio import Phylo
         tree = Phylo.read("datasets/opuntia.dnd", "newick")
         Phylo.draw_ascii(tree)
```

```
                             ________________ gi|6273291|gb|AF191665.1|AF191665
  ___________________________|
 |                           |   _______ gi|6273290|gb|AF191664.1|AF191664
 |                           |__|
 |                              |_____ gi|6273289|gb|AF191663.1|AF191663
 |
_|__________________ gi|6273287|gb|AF191661.1|AF191661
 |
 |___________ gi|6273286|gb|AF191660.1|AF191660
 |
 |    __ gi|6273285|gb|AF191659.1|AF191659
 |___|
     | gi|6273284|gb|AF191658.1|AF191658
```

13 Biopython - PSSM matrix

```
In [53]: from Bio.Seq import Seq
         test_seq=Seq("TAAGCGTGCACGCGCAACACGTGCATTA")
         test_seq
         print(test_seq)
```

```
TAAGCGTGCACGCGCAACACGTGCATTA
```

```
In [54]: from Bio import AlignIO
         from Bio.Align import AlignInfo
```

- Pfam은 단백질 페밀리 database, 각 서열 그룹을 align 한 파일이 제공됨
- Family: Sigma54_activ_2 (PF14532) https://pfam.xfam.org/family/PF14532#tabview=tab3

```
In [ ]: align = AlignIO.read("datasets/PF14532_full.txt", "stockholm")
        print(align)
        len(align)
        type(align)
        print(align[0])
        print(align[0].seq)
        print(align[0].format("clustal"))
```

- slicing alignment

```
In [56]: # print(align[3:8].format("clustal"))
         print(align[3:8,:200].format("clustal"))
         print(align[3:8,197])
```

```
CLUSTAL X (1.81) multiple sequence alignment


V7EPJ0_9RHOB/141-283            ----------------------------------------------------
B1ZTM1_OPITP/145-296            ----------------------------------------------------
W3ANH6_9FIRM/219-355            ----------------------------------------------------
Q6LNI3_PHOPR/144-289            ----------------------------------------------------
A0A1G8U4Y5_9RHOB/145-284        ----------------------------------------------------

V7EPJ0_9RHOB/141-283            ------------------------------VGRTPA-M-Q-A-L-Y-R-L-
B1ZTM1_OPITP/145-296            ------------------------------IGQSAS-M-R-K-L-V-Q-Q-
W3ANH6_9FIRM/219-355            ------------------------------y--KSRK-M-Q-K-T-V-D-L-
Q6LNI3_PHOPR/144-289            ------------------------------IGDSPL-S-V-K-L-R-E-Q-
A0A1G8U4Y5_9RHOB/145-284        ------------------------------r-GTSPQ-S-E-E-L-R-A-R-

V7EPJ0_9RHOB/141-283            --V---A---R---V---M--------N---T----D-------L-----
B1ZTM1_OPITP/145-296            --V---K---K---L---A--------A---V----R-------T-----
W3ANH6_9FIRM/219-355            --A---E---K---L---S--------R---T----D-------C-----
Q6LNI3_PHOPR/144-289            --I---A---N---I---A--------L---T----N-------K-----
A0A1G8U4Y5_9RHOB/145-284        --V---R---L---V---A--------R---A----G-------A-----

V7EPJ0_9RHOB/141-283            ---A-----V---L-V-T--GES-GT-GK----S----L-I-A----K--
B1ZTM1_OPITP/145-296            ---P-----V---L-L-I--GEN-GS-GK----S----A-V-A----E--
W3ANH6_9FIRM/219-355            ---P-----K---L-I-V--EPV-GN-LH----R----A-F-I----N--
Q6LNI3_PHOPR/144-289            ---D-----V---L-I-D--GES-GT-GR----R----T-V-S----K--
```

```
A0A1G8U4Y5_9RHOB/145-284            ---E-----V---L-V-T--GPT-GS-GT----A----K-V-A----E--


KENKE
```

- Turn the alignment object into an array of letters

```
In [57]: import numpy as np
         from Bio import AlignIO
         #align = AlignIO.read("PF05371_seed.sth", "stockholm")
         align = AlignIO.read("datasets/opuntia.aln", "clustal")
         align_array = np.array([list(rec) for rec in align], np.character)
         print("Align shape %i by %i" % align_array.shape)
         print(align_array)
```

```
Align shape 7 by 906
[[b'T' b'A' b'T' ... b'A' b'G' b'A']
 [b'T' b'A' b'T' ... b'A' b'G' b'A']
 [b'T' b'A' b'T' ... b'A' b'G' b'A']
 ...
 [b'T' b'A' b'T' ... b'A' b'G' b'A']
 [b'T' b'A' b'T' ... b'A' b'G' b'A']
 [b'T' b'A' b'T' ... b'A' b'G' b'A']]
```

```
In [58]: [rec for rec in align]
```

```
Out[58]: [SeqRecord(seq=Seq('TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLette
          SeqRecord(seq=Seq('TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLette
          SeqRecord(seq=Seq('TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLette
          SeqRecord(seq=Seq('TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLette
          SeqRecord(seq=Seq('TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLette
          SeqRecord(seq=Seq('TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLette
          SeqRecord(seq=Seq('TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLette
```

Note that this leaves the original Biopython alignment object and the NumPy array in memory as separate objects - editing one will not update the other!

```
In [59]: align_array.shape
```

```
Out[59]: (7, 906)
```

- SummaryInfo 클래스
- consensus sequence, position specific score matrix 계산
- information content와 substitution 정보 계산 가능

```
In [60]: summary_align = AlignInfo.SummaryInfo(align)
         consensus = summary_align.dumb_consensus()
         print(consensus)
         my_pssm = summary_align.pos_specific_score_matrix(consensus, chars_to_ignore = ['N', '-'])
         #print(my_pssm)
         # your_pssm[sequence_number][residue_count_name]
         print(my_pssm[1])
         print(my_pssm[1]["A"])
```

```
TATACATTAAAGXAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAAATGAATCTAAATGATATAXGATTCCACTATGTAAGGTCTTTGAATCATA
{'A': 7.0, 'C': 0, 'G': 0, 'T': 0}
7.0
```

```
In [61]: instances = [al.seq for al in align[:10]]
         print(instances)
```

```
[Seq('TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAA...AGA', SingleLetterAlphabet()), Seq('TATA
```

14 Biopython - Motif

- Bio.motifs package included in Biopython 1.61

```
In [62]: from Bio import motifs
         from Bio.Seq import Seq
```

```
In [63]: instances = [Seq("TACAA"),
                      Seq("TACGA"),
                      Seq("TACAA"),
                      Seq("TAGAA"),
                      Seq("TACAA"),
                      Seq("AACGA"),
                     ]
```

```
In [64]: m = motifs.create(instances)
         print(m)
```

```
TACAA
TACGA
TACAA
TAGAA
TACAA
AACGA
```

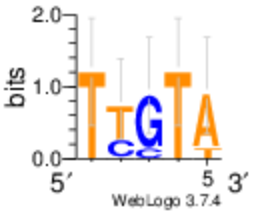
```
In [65]: m.counts
```

```
Out[65]: {'A': [1, 6, 0, 4, 6],
          'C': [0, 0, 5, 0, 0],
          'G': [0, 0, 1, 2, 0],
          'T': [5, 0, 0, 0, 0]}
```

```
In [66]: m.counts["A", 1]
         r = m.reverse_complement()
         print(r.consensus)
         #r.weblogo("mymotif.png")
```

```
TTGTA
```

```
In [92]: from IPython.display import Image, display
         display(Image(filename="mymotif.png"))
```



- Position-weight matrices 계산
- .counts 특성 사용

```
In [68]: pwm = m.counts.normalize(pseudocounts=0.5)
         print(pwm)
```

```
        0     1     2     3     4
A:   0.19  0.81  0.06  0.56  0.81
C:   0.06  0.06  0.69  0.06  0.06
G:   0.06  0.06  0.19  0.31  0.06
T:   0.69  0.06  0.06  0.06  0.06
```

```
In [69]: pssm = pwm.log_odds()
         print(pssm)
```

```
        0     1     2     3     4
A:  -0.42  1.70 -2.00  1.17  1.70
C:  -2.00 -2.00  1.46 -2.00 -2.00
G:  -2.00 -2.00 -0.42  0.32 -2.00
T:   1.46 -2.00 -2.00 -2.00 -2.00
```

```
In [70]: background = {"A":0.3,"C":0.2,"G":0.2,"T":0.3}
         pssm = pwm.log_odds(background)
         print(pssm)
```

```
        0     1     2     3     4
A:  -0.68  1.44 -2.26  0.91  1.44
C:  -1.68 -1.68  1.78 -1.68 -1.68
G:  -1.68 -1.68 -0.09  0.64 -1.68
T:   1.20 -2.26 -2.26 -2.26 -2.26
```

- 서열 내 모티프 존재 유무 탐색

```
In [71]: test_seq=Seq("TACACTGCATTACAACCCAAGCATTA")
         for pos, seq in m.instances.search(test_seq):
             print("%i %s " % (pos, seq))
         print(m)
```

```
10 TACAA
TACAA
TACGA
TACAA
TAGAA
TACAA
AACGA
```

```
In [72]: for pos, seq in r.instances.search(test_seq):
             print("%i %s " % (pos, seq))
         print(r)
```

```
TTGTA
TCGTA
TTGTA
TTCTA
TTGTA
TCGTT
```

- Using the PSSM score

```
In [73]: for pos, score in pssm.search(test_seq, threshold=3.0):
             print("%d, %f " % (pos, score))
         print(pssm.calculate(test_seq))
```

```
0, 3.643981
10, 6.759458
[  3.643981    -8.560285    -2.4004133   -5.6533937   -4.2748823
  -0.05645879 -10.145247    -3.3293302   -5.9753222   -3.5703382
   6.759458    -5.3903594   -5.8598447   -0.81545067  -0.81545067
   0.7695118   -6.3903594   -3.5379167    0.4255574   -1.9309279
 -10.145247    -3.3293302 ]
```

```
In [74]: m.pseudocounts = 0.1
         print(m.counts)
         print(m.pwm)
         print(m.pssm)
```

```
        0      1      2      3      4
A:   1.00   6.00   0.00   4.00   6.00
C:   0.00   0.00   5.00   0.00   0.00
G:   0.00   0.00   1.00   2.00   0.00
T:   5.00   0.00   0.00   0.00   0.00

        0      1      2      3      4
A:   0.17   0.95   0.02   0.64   0.95
C:   0.02   0.02   0.80   0.02   0.02
G:   0.02   0.02   0.17   0.33   0.02
T:   0.80   0.02   0.02   0.02   0.02
```

```
       0      1      2      3      4
A:  -0.54   1.93  -4.00   1.36   1.93
C:  -4.00  -4.00   1.67  -4.00  -4.00
G:  -4.00  -4.00  -0.54   0.39  -4.00
T:   1.67  -4.00  -4.00  -4.00  -4.00
```

15 기계학습 K-Nearest Neighbor (KNN)

```
In [75]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt

         from sklearn.datasets import load_breast_cancer
         from sklearn.metrics import confusion_matrix
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.model_selection import train_test_split
         import seaborn as sns
         sns.set()

In [76]: breast_cancer = load_breast_cancer()

In [77]: print(breast_cancer.data.shape)
         X = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)

(569, 30)


In [78]: print(X.loc[0:3,])

   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597

   mean fractal dimension  ...  worst radius  worst texture  worst perimeter  \
0                 0.07871  ...         25.38          17.33           184.60
1                 0.05667  ...         24.99          23.41           158.80
2                 0.05999  ...         23.57          25.53           152.50
3                 0.09744  ...         14.91          26.50            98.87

   worst area  worst smoothness  worst compactness  worst concavity  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
```

```
   worst concave points  worst symmetry  worst fractal dimension
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300

[4 rows x 30 columns]
```

```
In [79]: X[['mean area', 'mean compactness']]
```

Out[79]:

	mean area	mean compactness
0	1001.0	0.27760
1	1326.0	0.07864
2	1203.0	0.15990
3	386.1	0.28390
4	1297.0	0.13280
..	...	...
564	1479.0	0.11590
565	1261.0	0.10340
566	858.1	0.10230
567	1265.0	0.27700
568	181.0	0.04362

[569 rows x 2 columns]

```
In [80]: y = pd.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
         y = pd.get_dummies(y, drop_first=True)
         y
```

Out[80]:

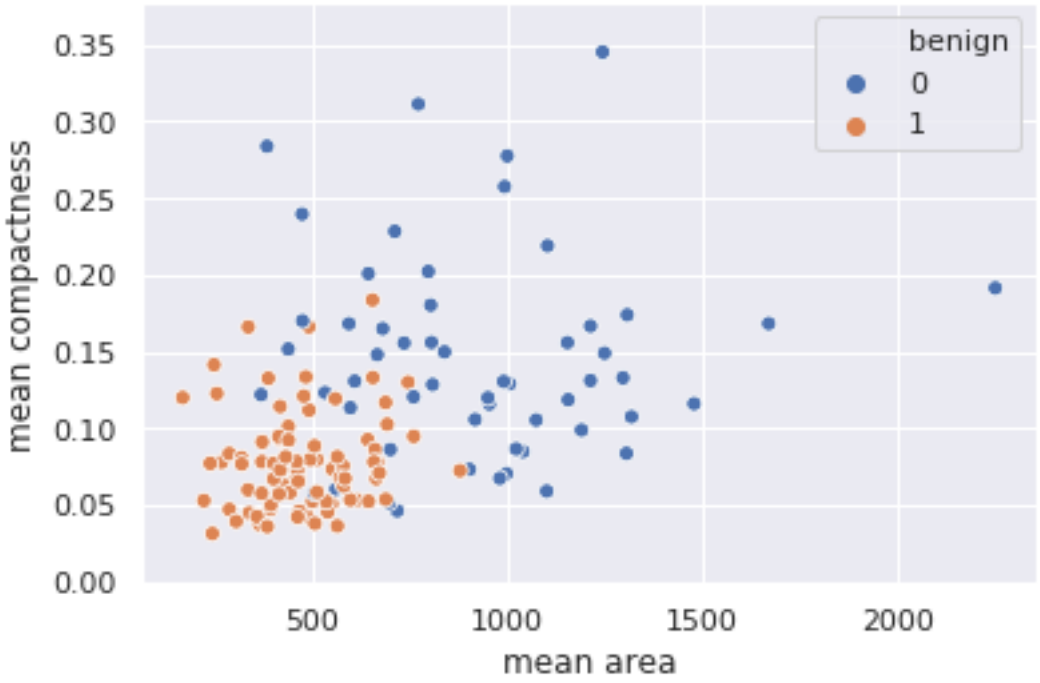
	benign
0	0
1	0
2	0
3	0
4	0
..	...
564	0
565	0
566	0
567	0
568	1

[569 rows x 1 columns]

```
In [81]: X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

In [82]: knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
         knn.fit(X_train, y_train)
         y_pred = knn.predict(X_test)
         sns.scatterplot(
             x='mean area',
             y='mean compactness',
             hue='benign',
             data=X_test.join(y_test, how='outer')
         )
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y
```

Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc718f20e90>



In [83]:
```
results = pd.DataFrame([y_test["benign"].values, y_pred], index=("test", "predict")).T
print(results)
```

```
     test  predict
0       1        0
1       0        0
2       1        1
3       0        0
4       0        0
..    ...      ...
138     1        1
139     1        1
140     0        0
141     0        0
142     1        1

[143 rows x 2 columns]
```

In [84]:
```
results[results["test"]+results["predict"] == 1]
```

Out[84]:

	test	predict
0	1	0

```
33      0       1
38      0       1
63      1       0
76      0       1
77      0       1
110     0       1
127     1       0
137     1       0
```

```
In [85]: from sklearn import metrics
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9370629370629371
```