##머신 러닝 공부

1. 기초 파이썬(계속 추가해 나가기)
   * 1. Enumerate: 주로 for 문과 같이 객체의 인덱스 값을 나열해주는 역할을 한다.

For index, value in enumerate(“abcd”): 로 할 시 tuple형태로 반환

* + 1. Lambda: 길게 쓰기 귀찮을 때 짧게 쓰는 역할을 한다.   
       m=(lambda x,y:x\*y)(10,20)이런식으로 바로 값을 넣을 수도 있고  
       mylist=[lambda a,b:a+b, lambda a,b: a\*b] 🡺mylist[0](3,4)처럼 꺼내먹을 수 있다
    2. Zip: zip(iterable1,2)로 두 iter을 차례대로 하나씩 묶어주는 역할을 한다
    3. Map,filter: 고약한 녀석들인데 함수,iter 형태이고 각각 iter를 함수에 대입하여 모든 함수 수행값을 리턴하거나 리턴값이 참인 것만을 묶어 돌려준다.  
       list(filter(lambda x:x>0,[1,-3,2,0]))   
       list(map(lambda x:x\*\*2,[1,2,3,4]))
    4. 문자열 다듬기   
       line=open('c:\\GH\\example.txt','r').readline()  
       line.strip().split(‘,’) 🡪사실 stip(‘,.’)처럼 쓰면 ,와 .를 없애준다. 즉 삭제할 문자들을 적어주는 것이다.  
       line.upper()/lower()  
       line.replace(‘a’,’c’)  
       line.count(‘a’) a의 개수를 반환 vs line.find(‘a’) a가 나타나는 첫 인덱스만 반환  
       ‘:’.join(line) or ‘-‘.join(list) 🡪list의 각 원소 사이에 해당 문자 – 를 끼어넣어 하나의 문자열로 만들어 준다.   
       ex) nums=[1,2,3,4]처럼 숫자로 된 것을   
       newnums=“ “.join(map(str,nums)) 처럼 각각 문자열화한다음에 이를 공백을 끼어넣어 하나의 문자열 1 2 3 4로 만들 수 있다.  
       list.index( value ) : 값을 이용하여 위치를 찾는 기능

list.extend( [value1, value2] ) : 리스트 뒤에 값을 추가 (‘+’연산자 보다 성능이 좋음)

list.insert( index, value ) : 원하는 위치에 값을 추가

list.sort( ) : 값을 순서대로 정렬

list.reverse( ) : 값을 역순으로 정렬

* + 1. Collections  
       Counter 에 대해 알아보자. Counter(iterable-list,string)을 입력하면  
       딕셔너리 형태로 {‘글자 하나하나’:2,’아니면 리스트 원소 하나하나’:3,}이런식으로 반환  
       주로 리스트를 넣어서 세는 식으로 할 것이다.  
       for I,v in Counter.items() 할 수도 있겠지만 for I in Counter된 것 하면 앞의 key로 정렬된다.
    2. A in b 구문  
       b가 무엇이냐에 따라서 많이 달라진다. 예를 들어 b가 문자열이라고 치면 a라는 문자열이 b안에 부분집합처럼 포함만되면 true를 뱉는다. In indigo-T   
       만약 b가 list 타입이라면 완전히 같은 것이 존재해야 true를 내뱉는다 in [‘indigo’]-F 만약 b가 dic 타입이라면 value가 완전히 같아서도 안되고 key가 완전히 같아야만 true를 내뱉는 까탈스러움을 보인다.
    3. Itemgetter  
       inventory = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]   
       map(itemgetter(1), inventory):[3, 2, 5, 1]   
       sorted(inventory-리스트, key=itemgetter(1) key=lambda x: -c[x]이렇게도 가능:[('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]

Itemgetter는 이처럼 튜플의 한 요소를 기준점으로 정렬하거나 뽑아낼 때 사용한다.  
sorted는 itemgetter로 얻은 애들을 기준으로 정렬할 때 사용한다.

* + 1. train\_test\_split  
       import sklearn from sklearn.model\_selection import train\_test\_split  
       a, b = np.arange(10).reshape((5, 2)), range(5)  
       train\_test\_split(a,b,random\_state=1)   
       이때 X\_train, X\_test, Y\_train ,Y\_test 와 같은 방법으로 initialization을 한다.  
       이때 random\_state 는 매번 랜덤으로 나오는 결과 값을 하나의 메모리에 저장시켜서 그 숫자를 입력시 결과가 늘 일정하게 나오게 해주는 역할을 한다.
    2. Astype란 뭘 하는 놈일까?  
       x = np.array([1, 2, 2.5]):array([ 1. , 2. , 2.5])🡪이런식으로 하나라도 float 존재시 그에 맞춰서 저장된다. 이런 경우 문제가 발생할 수 있기 때문에

x.astype(int):array([1, 2, 2])🡪 이처럼 array안의 배열을 모두 원하는 타입으로 바꿔 주는 역할을 하게 된다.

* + 1. file을 읽고 써보자..!  
       file=open(‘c:\\GH\\hello.txt’,’w’)이는 쓰기 모드로 연다. 이때 파일이 없다면 새로 생기면서 열리게 된다. 그런데 이는 파일을 단순히 여는 것이고 실제로 글을 쓰려면 file.write(‘~~~’) 이런식으로 써야한다. 이를 조금 더 효율적으로 써보자.  
       with open(‘c:\\~~’,’w’,encoding=’UTF8’) as f: 이때 지명을 하면서 밑의 것을 이어서 하는 커맨드이고 이는 주로 리스트의 내용들을 쓰는 방법으로 사용된다.  
        for item in list:  
        f.write(“%s\n” % item) 이때 list 속 item은 모두 문자열이어야하고 하나씩 할 떄마다 한줄 씩 띄는 식으로 코딩하는 것 C언어와 비슷한 부분이다.   
         
       읽는 식은 조금 복잡해 보인다만 단계별로 살펴보자.   
       1. with open('hello.txt', 'r') as file: # hello.txt 파일을 읽기 모드(r)로 열기

lines = file.readlines()🡪readlines는 각 파일을 한줄씩 읽어서 리스트로 저장해주는 직관과는 조금 거리가 먼 친구이다. ['안녕하세요.\n', '파이썬\n', '코딩 도장입니다.\n']  
🡪이런식으로 한줄씩 \n을 넣어 리스트에 저장해준다.

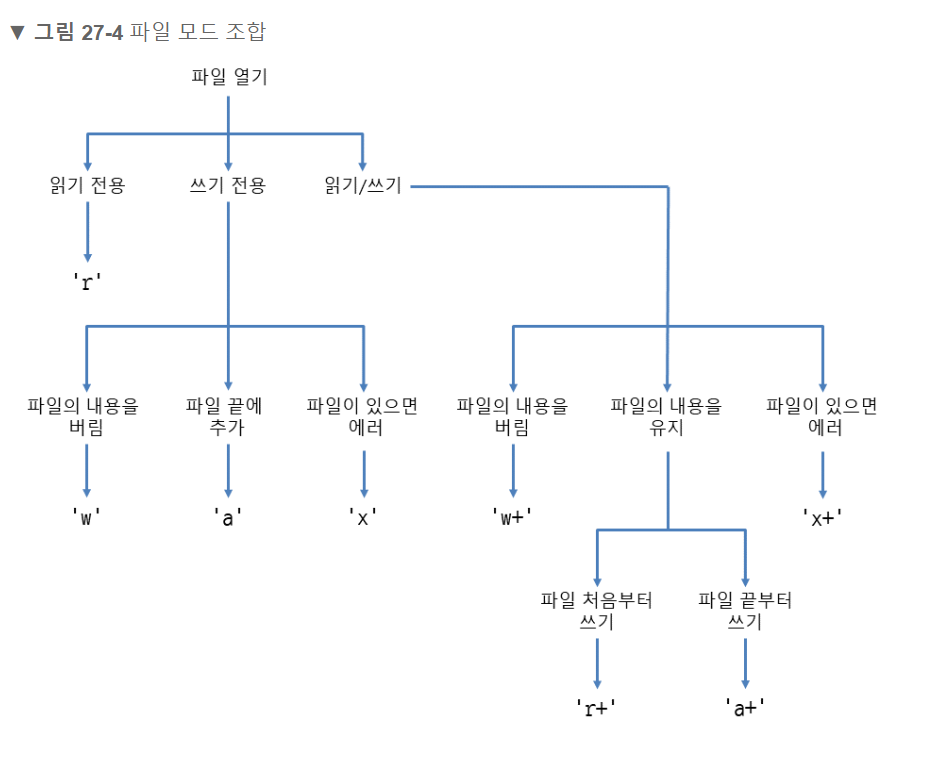
2. with open('hello.txt', 'r') as file: # hello.txt 파일을 읽기 모드(r)로 열기

line = None # 변수 line을 None으로 초기화(그래야 while문에서 비교가 가능)

while line != '': #readlines와 달리 readline은 한줄씩 읽는다. 끝까지 알아서 되는게 아니기 때문에 반복문을 사용해야한다. 그런데 아무것도 읽을 게 없어지면 ‘’의 빈 문자열을 반환하기 때문에 이것과 같지 않으면 계속 출력하게 해주는 것이다.

line = file.readline()

print(line.strip('\n')) #이때 \n도 붙어 있기에 삭제하여 출력(print에 띄어쓰기 이미 자동포함되어 있기 때문에)

3.   
with open(‘’,’r’,encoding=’utf8’) as f:  
 for text in f: 🡪파일을 한 줄씩 읽어서 text란 변수에 저장해준다. 이때 readline의 전형적인 특성처럼 각 문자열에 \n이 포함된다는 것 주의  
 for tweet in text.split(‘\n’):🡪이는 2가지 의미가 이는데 각 줄 안에서도 또 여러 트윗들이 존재하기 때문에 그 것을 각각 \n으로 나누어 tweet에 저장해주고 동시에 각 줄의 끝에 있는 \n까지 자동적으로 걸러지는 부가 효과가 있다.  
그런데 사실 이는 사실 줄 단위로 읽는 readline에 \n이 포함되어 있지 않을 것이기 때문에 youtube.append(text.strip(‘\n’)) 으로 한번에 하는 것이 훨씬 낫다.   
이렇게 읽을 때는 읽은 것을 쥬피터 안의 lst에 넣는 방식으로 활용하는 것이다.   
  
혹시 변수를 파일에 저장하고 불러오는 방법을 알고 싶으면 pickle에 관한 <https://dojang.io/mod/page/view.php?id=2327> 링크를 참고해보도록 하자.

* + 1. 함수에 대해 조금 자세히 알아보자..!  
       먼저 함수의 다중 return 이는 c언어와 다른 부분으로 편의성 개선이다.  
       def \_\_(a,b):  
        return a+b,a-b 🡪 이런식으로 하면 결과값이 (1,2) 이렇게 튜플형식으로 지정된다. 파이썬에서 , 로 구분되는 것은 자동 튜플 지정된다. 언패킹을 이용해 변수 지정을 동시에 x,y=\_\_(a,b)이런식으로 하기에 유용.

다음 방법은 함수에 넘겨주는 인자를 매번 변하게 할 수 있는 매우 유용한 기법임.  
def print\_numbers(a,\*args): 🡪\*는 리터레이블을 하나씩 풀어주는 역할을 한다.

이런식으로 쓸 경우 앞은 고정인자 뒤에는 변하는 가변 인자로 할 수 있다.   
... print(a)

... print(arg)  
>> print\_numbers(1) 🡪이 것처럼 뒤의 가변 인자에 아무것도 안 넣을 수도 있다.

1

()

>>> print\_numbers(1, 10, 20)

1

(10, 20)

>>> print\_numbers(\*[10, 20, 30]) 🡪풀려서 10,20,30으로 들어가고 가변 부분 , 형태이기에 튜플로 출력

10

(20, 30)  
  
이번엔 dictionary를 언팩킹해서 가변 인자로 하는 방법을 알아보자. 기존 처럼 \*를 한번만 하면 key값이 들어가게 되고 \*\*를 두번 해야 value 값으로 들어가게 된다.   
>>> x = {'name': '홍길동', 'age': 30, 'address': '서울시 용산구 이촌동'}

>>> personal\_info(\*x)

이름: name

나이: age

주소: address

>>> x = {'name': '홍길동', 'age': 30, 'address': '서울시 용산구 이촌동'}

>>> personal\_info(\*\*x)

이름: 홍길동

나이: 30

주소: 서울시 용산구 이촌동

>>> def personal\_info(\*\*kwargs):🡪여기서 주의해야할 점은 kwargs는 \*\*로 풀기 전 그 자체인 딕셔너리를 의미하기 때문에 딕셔너리로 봐야한다는 것이다. 그리고 가변인자로 들어오는 값은 age=30,addres=’—‘처럼 풀어져 있는 애가 들어간다는 점이다. 이를 딕셔너리로 합쳐서 알아서 이해하는 것이다.

... for kw, arg in kwargs.items():

... print(kw, ': ', arg, sep='')

...

* + 1. 모듈을 import해보자..!

>>> import math

>>> math.sqrt(4.0)

2.0

1. >>> import math as m    # math 모듈을 가져오면서 이름을 m으로 지정
2. >>> m.sqrt(4.0)         # m으로 제곱근 함수 사용
3. 2.0
4. >>> from math import \*    # math 모듈의 모든 변수, 함수, 클래스를 가져옴
5. >>> pi
6. >>> from math import pi as p, sqrt as s
7. >>> p         # p로 원주율 출력
8. 3.141592653589793
9. >>> s(4.0)    # s로 sqrt 함수 사용
10. 2.0

그냥 math 모듈을 import해오면 math.sqrt처럼 귀찮게 앞에 다 붙여야 되고 \*로 전부가져오면 그냥 막 써도된다. 특정 부분pi,sqrt만 가져와서 이를 약자로 정해서 쓸 수도 있다.

>>> import urllib.request

>>> response = urllib.request.urlopen('http://www.google.co.kr’)

🡪원래 이런식으로 패키지.모듈.함수를 불러와야하는데 사실상 .을 두번 치는게 매우 귀찮다.

>>> import urllib.request as r    # urllib 패키지의 request 모듈을 가져오면서 이름을 r로 지정🡪그나마 조금 간소화 가능

>>> response = r.urlopen('http://www.google.co.kr')    # r로 urlopen 함수 사용

>>> from urllib.request import Request, urlopen    # urlopen 함수, Request 클래스를 가져옴(from urllib.request import \* 해도 같다)

>>> req = Request('http://www.google.co.kr')       # Request 클래스를 사용하여 req 생성

>>> response = urlopen(req) 🡪urlopen에 직접 주소 안 넣고 주소 넣어서 초기화한 인스턴스를 넣는다.또한 Request 와 urlopen 모두 as 로 간소화 가능하다는 것 유념.

* + 1. 예외 처리를 알아보자

try:

    index, x = map(int, input('인덱스와 나눌 숫자를 입력하세요: ').split())

    print(y[index] / x)

except ZeroDivisionError as e:                    # as 뒤에 변수를 지정하면 에러를 받아옴

    print('숫자를 0으로 나눌 수 없습니다.', e)

보통 이런 식으로 쓰고 except Exception as e를 사용하면 모든 예외 처리 가능. Exception이 모든 예외 클래스의 parent class격이기 때문.

try:

    x = int(input('나눌 숫자를 입력하세요: '))

    y = 10 / x

except ZeroDivisionError:    # 숫자를 0으로 나눠서 에러가 발생했을 때 실행됨

    print('숫자를 0으로 나눌 수 없습니다.')

else:                        # try의 코드에서 예외가 발생하지 않았을 때 실행됨

    print(y)

finally:                     # 예외 발생 여부와 상관없이 항상 실행됨

    print('코드 실행이 끝났습니다.')

#사용자가 직접 정의하는 에러

class NotThreeMultipleError(Exception):    # Exception을 상속받아서 새로운 예외를 만듦

    def \_\_init\_\_(self):

        super().\_\_init\_\_('3의 배수가 아닙니다.')

def three\_multiple():

    try:

        x = int(input('3의 배수를 입력하세요: '))

        if x % 3 != 0:                     # x가 3의 배수가 아니면

            raise NotThreeMultipleError    # NotThreeMultipleError 예외를 발생시킴

1. CLASS

이번에는 클래스에 대해 알아보자!  
  
class는 파이썬의 객체 지향의 근본이 되는 문법이다. 간단한 모양을 살펴보자.  
class person:  
 def \_\_init\_\_(self, input variable –a ,b):🡪각 인스턴스를 input 값을 할당하여 초기화 시켜주는 매직 method이다. 이 a,b는 person(a,b)로 쓰일 것. Self는 현재 인스턴스 자신을 넣는다 보면 편하다.

(이때 def \_\_init\_\_(self,\*args): self.name=args[0] 이런식으로 가변 입력 값을 받을 수 있다. 딕셔너리 형태의 가변 인수를 받고 싶으면 def \_\_init\_\_(self,\*\*kwargs): 하면 되나 self.name=kwargs[‘name’]으로 클래스를 짤때부터 입력 값으로 주어지는 딕셔너리의 key의 이름을 알고 작성해야하기 때문에 리스트 형태의 가변 인수를 선호하는 편이다.)  
 self.attribute\_name=a🡪self.특정 속성에 input 값을 할당하는 형태이다.   
 self.tax\_fee=그 나라 tax율

이때 조금 헷갈릴 수 있지만 instance.attribute\_name은 속성을 불러 오는 친구고   
instance.method\_name으로 쓰면 메소드 대신 속성을 검색한다. 따라서 instance.method\_name(\*args)이런식으로 해줘야 메소드에 접근 할 수 있다.

def method\_name(self, input variable –c,d)🡪이는 person.method(c,d)로 method 자체에 넣어주는 variable을 의미한다.

if c\*self.tax\_fee >=self.\_\_wallet(비공개 속성):

그런데 클래스에 미리 설정된 속성 말고 내가 추가를 하고 싶다. (메소드를 추가하고 싶다면?) 그러면 그냥 instance.원하는\_속성\_이름=aa 이런 식으로 추가할 수 있다. 단, 이렇게 추가한 인스턴스의 속성은 그 인스턴스에게만 유지된다는 점을 기억해야한다.   
그런데 이를 악용하여 정해진 클래스의 용도를 벗어나게끔 인스턴스에 속성을 추가하는 것을 막기 위해서 slots이라는 속성 제한이 있다. 이는 메소드가 아니고 그냥 그 클래스 자체의 속성이라 보는 것이 좋다.

Class person:  
\_\_slots\_\_=[‘attribute1’]  
bag=[]  
def put\_bag(self,stuff):  
 person.bag.append(stuff)🡪그런데 파이썬에서는 james=person() 하고 james.bag하면 \_\_init\_\_안에 있는 인스턴스의 속성을 먼저 찾고 거기에 없으면   
클래스 person자체의 속성을 찾는다. 따라서 겉보기에는 인스턴스 속성인 것 같지만 클래스 속성인 것이다.

여태껏 인스턴스 속성과 클래스 속성에 대해 알아보았으니 이제 인스턴스 메서드와

클래스 메서드 등을 알아보자.

먼저 인스턴스 메서드는 우리가 지금까지 배워왔던 것이다. 그렇다면 클래스 메서드

와 정적 메서드는 왜 필요한 것일까?

이야기를 하기에 앞서서 각 메서드의 형태를 알아보자.

<정적메서드>

Class Calc:

@staticmethod🡪decorator이며 메서드에 추가 기능을 구현할 때 메서드 위 사용

Def add(a,b):🡪가장 중요한 것은 self를 받지 않는다. 즉, 인스턴스의 속성에 접근 하지 않는다는 것이다. 따라서 이 메서드를 실행하여도 외부(인스턴스)에 영향을 끼치지 않는 순수 함수pure function이고 부수 효과side effect가 없다. 즉, 입력 값이 같으면 언제나 같은 값을 반환하게 된다.

>>> a = {1, 2, 3, 4}

>>> a.update({5}) # 인스턴스 메서드

>>> a

{1, 2, 3, 4, 5}

>>> set.union({1, 2, 3, 4}, {5}) # 정적(클래스) 메서드

{1, 2, 3, 4, 5}

<클래스 메서드>

class Person:

count = 0 # 클래스 속성

def \_\_init\_\_(self):

Person.count += 1 # 인스턴스가 만들어질 때 초기화가 이루어진단 것을 이용하여 클래스 속성 count에 1을 더함

@classmethod

def print\_count(cls):🡪cls자리를 모두 Person으로 바꾸어도 같은 결과 도출

print('{0}명 생성되었습니다.'.format(cls.count)) # cls로 클래스 속성에 접근

james = Person()

maria = Person()

james.print\_count()

Person.print\_count() # 2명 생성되었습니다.

🡪클래스 메서드는 정적 메서드처럼 인스턴스 없이 클래스.메서드 하면 호출할 수

있다는 점은 같다. 하지만 정적 메서드가 아무것도 접근안하는 것에 비해 클래스 메

서드는 같은 클래스로 인스턴스가 몇 개 생성되엇는 지를 알기 위해서 같이

클래스 속성, 클래스 메서드에 접근해야 할 때 쓴다.

다음은 클래스 상속에 대해서 알아보자. 클래스 상속은

Class Person(parent class)와 같이 쓰고, 부모 클래스를 그대로 사용할 수 있다.

Issubclass(Person,parent class)하면 관계를 확인할 수 있다. 이런 클래스 상속은 동일

한 클래스 내용을 중복해서 쓰는 수고를 덜어줄 때 유용하게 쓰일 수 있다. 이와 같

은 상속 관계는 두 클래스가 같은 종류이고 동등한 관계일 때 사용한다. 하지만

class Person:

    def greeting(self):

        print('안녕하세요.')

class PersonList():

    def \_\_init\_\_(self):

        self.person\_list = []    # 리스트 속성에 Person 인스턴스를 넣어서 관리

    def append\_person(self, person):    # 리스트 속성에 Person 인스턴스를 추가하는 함수

        self.person\_list.append(person)

다음과 같이 클래스 상속이 아니라 새로운 클래스의 인스턴스 속성에 Person으로 만들어진 인스턴스를 추가하는 방식으로 사용된다면 이는 포함관계라고 할 수 있다.

class Person:

    def \_\_init\_\_(self):

        print('Person \_\_init\_\_')

        self.hello = '안녕하세요.'

class Student(Person):

    def \_\_init\_\_(self):

        print('Student \_\_init\_\_')

super().\_\_init\_\_() 🡪이는 기반 클래스의 \_\_init\_\_메서드를 호출해서 실행해줌.

        self.school = '파이썬 코딩 도장'

james = Student()

print(james.school)

print(james.hello)    # 기반 클래스의 속성을 출력하려고 하면 에러가 발생함

이런 코드는 에러가 발생한다. Why? 이런 문제는 파생클래스(child class)로 기반 클래스(parent class)의 ‘인스턴스 속성’에 접근하려 할 때 발생한다. 물론 파생 클래스가 기반 클래스를 물려받기는 하지만 파생클래스로 인스턴스를 만들 때 기바 클래스까지 인스턴스가 만들어지면 안되기 때문에 기반 클래스 init의 초기화가 이루어지지 않는다. 따라서 그런 상황에서 기반 클래스의 인스턴스 속성을 사용할 수 없는 것이다.

이런 상황에서는 super(파생 클래스, self)를 사용한다. 이는 파생클래스의 기반 클래스의 메서드(꼭 init이 아니어도 된다)를 부르기 위해서 사용한다. (참고로 먼저 파생 클래스의 인스턴스 속성에 hello가있는 지 찾고 없으면 super로 부모 클래스의 init이 호출되었는 지 여부를 검토한다음에 결과를 내보낸다.)

사실 super는 init 외에도

class Person:

    def greeting(self):

        print('안녕하세요.')

class Student(Person):

    def greeting(self):

        super().greeting()    # 기반 클래스의 메서드 호출하여 중복을 줄임

        print('저는 파이썬 코딩 도장 학생입니다.')

james = Student()

james.greeting()

다음과 같이 메서드 오버라이딩을 할 때 원래 기능에 새로운 기능을 덧붙이는 용도로 사용한다. greeting으로 메서드 이름이 같을 때는 아예 새롭게 덮어쓰는 방식으로 파생클래스 안에서 메서드가 만들어지기 때문에 super를 써서 중복되는 부분은 따로 가져오는 식이다.

예외적으로 파생클래스에서 아예 init메서드가 없다면 파생 클래스 인스턴스를 만들 때 호출될 init이 없기 때문에 자동적으로 부모 클래스 init을 끌어오기에 이때는 super로 초기화해주지 않아도 된다.

다중 상속을 할 때 그 상속 순서를 확인하고 싶으면 Method Resolution Order-MRO를 사용하면 된다.   
>>> D.mro()

[<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class 'object'>]

이렇게 순서를 쉽게 볼 수 있는데 object클래스라고 모든 클래스의 조상인 클래스가 항상 맨 뒤에 붙는다는 것을 유의하자. 모든 클래스는 object클래스를 상속 받기 때문에 class x == class x(object)와 같다.

1. Numpy

Array의 선언  
arr=np.array([1,2,3]) np.arange(10)   
np.zeros((5,7) :2번 괄호로 튜플로 shape형태 지정 np.full((2,3),5) np.eye(3) np.random.randint(0,3,(6,6))

Array의 매서드  
.reshape((i,j)) .reshape((-1,3)) :열이 3이 되게 행을 알아서 조절   
.ndim .size .shape

idx = np.array(idx, dtype=bool) 이는 idx를 bool 타입으로 바꾸어주는 방법  
np.random.randint(0,2,(6,6),dtype=bool) array==2 같은 식으로 하면 bool type 나옴  
array[bool\_idx] 하면 true에 해당하는 자리 원소들만 array 형태로 반환

기타  
np.linspace(a,b,x) 하면 a이상 b 이하의 범위에서 x개만큼을 일정한 간격으로 찍어주는 아주 편리한 일차원 어레이를 만들 수 있다.

1. Pandas
2. DataFrame 형성  
   temp={“1열이름”:[“1열 원소”,”a”],”2열 이름”:[true,false]} base 자료형은 딕셔너리 형태  
   df=pd.DataFrame(temp,[‘index1’,’index2’])
3. DataFrame 슬라이싱  
   df[[‘1열이름’,’2열이름’]] [[]]처럼 두번해줘야한다는게 포인트  
   df.iloc[2,3] 원하는 포인트 데이터 접근 가능  
   df.loc[‘인덱스2’,’열1’] 이런식으로도 가능   
   df2= …. 똑같이 만든 다음에 df=df.append(df2) 이런식이 제일 깔끔  
   df.reset\_index().iloc[:,1:] 일종의 콤보. 기존 index를 새로운 열로 추가 후 제외한 것  
   df[df[‘열이름’]==False] 처럼 가능
4. DataFrame 주요 매서드  
   df[‘열이름’].mean() var,std,min,max  
   df[‘열이름’].apply(lambda x:x/10) 의 함수를 적용  
   df.head()  
   df.groupby([‘열이름’]).groups[2] df.groupby([‘분류할 열이름’])[‘열의 각 동일한 값에 대해 분류될 다른 열’].mean()
5. Matplotlib
6. Line chart  
   plt.plot(x축,sorted(원래 y축,reverse=True), color=’green’,marker=’x’,linestyle=’dashed’,linewidth=2,label=’wage’)  
   이때 plt.plot(x,f(x))처럼 y에 x대입한 함수 꼴로도 쓸 수 있다.  
   plt.title(‘차트이름’) plt.xlabel(‘x축 이름’) plt.xlim(a,b) 범위 제한   
   plt.grid(True) 격자표시   
   plt.legend() 스티커처럼 범례를 표위에 표시
7. Bar chart  
   x축이 문자열=[‘a’,’b’,’c’] xs=[I for i,\_ in enumerate(x축)] 처럼 하고   
   plt.bar(xs,y축) plt.xticks(xs,x축)
8. Histogram  
   plt.hist(df[‘열이름’,bins=20]) hist는 한 개의 열을 표로 높낮이 보여주는 친구.
9. Scatter plot  
   plt.scatter(df[‘x축이될 열’][:10],df[‘y축이 될 열’]]

for label로 하고 싶은 데이터,x,y in zip(df[‘년월일’],df[],df[]):  
 plt.annotate(label,xy=(x,y)) 이런식으로 각 점 라벨 붙임

1. Subplot 잡기술  
   plt.figure(figsize=(너비,높이))  
   plt.subplot(121) 1\*2로 쪼갠뒤 1번째 차트에 그리겠다는 것  
   그 후 plt.hist()등   
   plt.subplot(122)
2. Bubble chart
3. Overfitting and underfitting
4. 오버피팅이란?  
   모델의 성능이 학습 데이터에서는 좋지만 새로운 정보에 대해서는 잘 예측을 못할 때  
   이 경우는 핵심 요소가 아닌 다른 요소까지 중요 변수로 포함되어 계산되거나 데이터의 잡음까지 모델에 학습되어 그렇다.   
   이때 variance는 커지고 bias는 작아지는 경향이 있다.
5. 언더피팅이란?  
   모델의 성능이 학습데이터에서도 좋지 않은 경우를 의미한다. 즉, 중요 변수를 누락하고 식을 만들었단 것이다. 이 경우 variance는 낮고 bias는 높아진다.
6. 그렇다면?  
   그런데 데이터가 늘수록 variance가 감소하기 때문에 오버피팅은 줄어든다. 하지만 데이터가 늘수록 bias는 작아지지 않는다. 즉 데이터가 많아져도 중요 변수를 누락한 언더피팅은 나아지지 않는단 것이다.
7. 데이터 사이언티스트의 강의  
   Q&A  
   🡪시급, 추가 매출에 대한 인센티브, 전체 집행 비용의 몇 퍼센트, value added vc 주식을 싸게 달라. 등으로 협상 할 수 있음  
   경영 승계의 여론 조사. 20개의 sns를 모두 긁었다. 일베 등도 포함.   
   야구장 가면 우호적 댓글 🡪 야구장을 많이 가게 되었다.  
   what is a product manager  
   nc soft 등에서 일했고 noom 에서 일했다.  
   중요한 능력은 무엇일까. 영업력과 매출을 만드는 능력이 중요하고 풀 타임은 남들을 설득 하는 능력이 가장 중요한 것 같다. 구현화할 수 있는 프런트 백 엔드 등은 다 자잘자잘한 일들이 된다. 더 비싼 문제를 풀수록 더 비싼 몸값이 된다. 1.5(2.0 업계 평균이상)  
   데이터 분석가-연봉7억 카카오페이지에서 엄청난 매출을 만드는 사람이 있긴 하지만 아까 말했던 분야들 하나 하나 전문가가 되어도 무방하긴 하지만 궁극적으로 승진을 하고 연봉이 오를려면 결국 사람 설득하는 일이 가장 중요하게 된다. 따라서 사람들 설득시키는 일을 두려워 하지 마라. 신규 입사자로 활동을 하려면 업계에 따라 다르지만 데이터를 많이 쓰고 있으면 머신러닝을 잘 다룰지 알면(전체 퍼넬에 취소율 구매전환율 등을 뽑을 수 있다 이정도 kagg 많이 풀어보라)~~. Ux에 근거한 실제로 데이터를 발생시켜 그 데이터 기반으로 가설을 검증한 경험이 있으면   
   미래부 software maestro 등 정부 주도 프로그램을 아주 잘활용하면 인프라 장비를 아낄 수 있다. 아마존 구글에서 하는 competition을 자주 나가려 한다.   
   우버 air bnb 같은 큰 곳은 data scientist가 매우 중요한데 지금 막 (목표를 주면 실행한다) 스타트업을 시작하는 곳에서는 큰 부담이 될 것.   
   a.b test는 외부요인이 매우 많은 상황에서 매우 불완전한 접근 방식이다. 그나마 대조군을 비교할 수 있어서 하는 것일 뿐이다. 보통 market research를 많이 한다. Mab 방법론이 많이 활성화되어 있으니 참고.   
   우선순위 평가하기 근거 기반 idea가 중요 사람 설득하기(나혼자 노력해서 하는 일보다 결국에는 자원이 성장을 많이 만들어온다. 100만원 220만원으로 만드는 것보다 100억 을 110억으로 만드는게 훨씬 크단 것) 회사에서 그로스 팀이 이렇게 중요하니 자금을 더 달라.   
   아직까지 우리나라에 그로스해커를 경험 많이 쌓은 사람은 없어서 지금은 원하는 대로 다한다. 근거기반 가설 실험을 통해 매출을 얼만큼 올렸다는 한 두개의 사례만 있어도 가치가 오른다. 포트폴리오, 사고 과정을 잘 정리하는 법을 익혀 둬야한다. 다른 사람이 보기에도 말이 되는 그런 프로세스가 협상에 잘먹히고 가장 중요한 게 돈을 얼마나 벌어주는 지 여부  
   goodui에 들어가면 ui탬플릿 중 몇 개 나누는게 전환율이 좋냐 등을 갖고 토론하다가 abtest 결과 들을 모아놓은 것이 goodui이 이다.   
   자기주도 학습이 가능하다면 그 돈으로 차라리 본인의 동기부여를 유지하는 데 투자하라. Boot camp vs college 에 많은 토론이 있다. 자기주도 학습이 안되있는 사람들을 위해 ds school 같은 것이 있는 것.   
   매출을 만들어나가는 일 자체가 매우 재밌다.   
   기득권 있는 곳일수록 더 그로스 해킹으로 인한 솔루션 제시가 잘 이루어지지 않는다. 제러드 와인버그 퀄리티 소프트웨어 매니징 엔티시페이ㅣ징 체인지 부분 추천. 설득관련된 책의 리스트를 줄 것.   
   데이터가 짱이다 라고 하는 순간 싸움이 벌어진다. 갈등을 만드는 게 아니라 좋은 의사결정을 만들 수 있도록 서포팅 하는 일이다. 그래서 보통 컨설팅 조직 쪽으로 가서 다른 사람들의 성공할 수 있는 그런 조직으로 가는게 좋을 것.   
   동료들로 하여금 자기를 좋아하게 만들어라.   
   ac2.co.kr 가면 다양한 협업 환경과 논리를 만드는데 투자를 많이 했다. 관계,대화와 설득 부분을 가서 살펴보면 좋을 것이다.   
      
   growth hacker?  
   🡪제품 영역도 손에 댄다는 점에서 기존 마케터와 조금 다르다. 매출이 목표가 아니라 성장을 위한 여러 단계들을 하나씩 쌓아가는 작업. 요즘 마케터 중 performance   
   monthly paying user가 오르고 있으면 적자여도 ㄱㅊ. 한달간 20%정도 성장해야 성장한다고 평가받음. 그래서 기존방법이 아닌 그로스 운동이 탄생. 크로스펑셔널한 조직 구성 방법이 유행, ‘근거있는’ 가설을 만들고 이 가설을 투명하고 합리적인 방식으로 우선순위를 평가하여 실험을 돌리며 검증.  
     
   tip for starter  
   🡺 어디에서 새는 지를 알고 어떤 것을 우선순위로 고쳐갈지가 중요. 즉 우선순위를 선정해나가는 것이 효율적 활동의 가장 큰 것. / sandbox로 테스트를 해본다. selective하게 특정 사용자에게 국한하여 실험 영역을 만들 수 있는가가 중요. 사용자 모두에게 굳이 할 필요는 없다. /다양한 사람들의 idea를 제대로 평가하는 방법이 없다면 사회적 영향력이 쎈 사람 순서대로 하게 된다. 빠른 성장과 책임이라는 이유 때문에 스타트업이 오히려 심하다. 그래서 투명하고 합리적인 근거 기반으로 impact(구매전환율 등 의미가 큰 것을 바꾸는 가) confidence(근거와 확신이 있는 것) Ease(얼마나 이행하기 가는한가) 963 6610 의 차이는 생각보다 어마어마하다. 조그맣고 쉬운 것 하나하나 빨리 이행해서 단계 단계 성장해 나가는 것이 중요한 것. (레진 코믹스 90%여성인 걸 빠른 배움을 얻었기 때문에 그 후 방향을 다 바꿔나갈 수 있는 것이다)/ a.b test 언제 무슨 기준으로 끝내는가 이 부분이 생각보다 매우 중요하다. (박스앤위스커 박장시-전문지식 굳) 1번째 관문인 90-95%의 통계적 유의성을 얻으면 2 번째 관문인 샘플 사이즈를 본다. (Baseline conversion minimum detectable effect 을 각각 10,42%라면 1400명 정도 필요) 3번째 관문은 사용자의 라이프 싸이클을 보는 것이 중요하다. (70% 30대 직장 여성이 고객. 주중은 생활패턴 잘 지켜지다가 주말에 달라짐. 따라서 만약 주중과 주말은 다른 패턴을 보일 것이란 것/ 요식업 사장을 대상으로 하는 광고의 효과가 1.5배나 됨. 사장이 비와서 심심하면 인스타 봐서 광고 노출이 잘됨. )

사례들  
🡺 noom은 맞춤 사례를 주는 경우가 매우 중요하다. 내가 따라할 수 없는 모델을 보여주면 오히려 구매전환율이 떨어진다. 맞춤 경험이 유의미함. 0원으로 시작->천원으로 시작  
blank 대세라고 광고. 몇 명이 보고 있어요  
야나두 128기 17시간 등 마음의 동요를 노림 but 이런 것들은 고객 채널에 대한 정확한 이해를 바탕으로 해야하는 것이다.

기타 추천들  
🡺hackinggrowth.co.kr / jungsungyoung진화된 마케팅 그로스 해킹

1. 배울 수 있는 유용한 링크들   
   세계지도 https://rabernat.github.io/research\_computing/intro-to-basemap.html
2. 정규 표현식에 대해 배워보자
3. “.”에서 . 은 \n을 제외한 모든 문자  
   “+”는 1회 이상 반복. 이거 안쓰고 그냥[a-z]하면 [‘a’,’b’,’c’]등으로 끊겨서 단어 나옴  
   why?[]는 문자열 안의 것이 나오자마자 그 문자를 매칭시켜주기 때문에 ‘abc’를 세개의 원소들로 뱉고 +는 []안의 것이 한번 이상 등장하면 바로 잡아버린다(혹은 abc를 3번 뽑은 다음에 중복되니깐 하나로 축약)  
   {m,n}은 바로 앞의 문자가 m에서 n번 반복해서 등장할 때 매치  
   [^]와 달리 ^a는 a로 시작하는 문자열과 매치 a$는 끝부분  
   a?는 앞의 문자가 0 혹은 1 번만 등장-의외로 쓰임새가 있다  
   ab\*하면 앞의 문자가 0번이상 등장과 매치이므로 a,ab,abbb 등과 매치  
   [0-9] 숫자 중 하나라도 일치시 매치 [a-z0-9] 이런 식으로 붙여쓴다 [^]는 해당 문자에 포함되지 않으면 매치 시켜준다. \d 숫자와 매치 가능 이때 01x-xxxx-xxxx의 전화번호와 매치되는 문자열을 찾고 싶다? 어떡하면 좋을까?  
   01[0-9] 이런식은 너무 귀찮으니깐 01\d-\d{3,4}-\d{4}요렇게 표현  
   비슷하게 공백 문자를 단축하는 \s와 공백 문자 여집합인 \S가 있다.   
   \w로 alphanumeric 문자와 매치.(숫자, 알파벳, 한글 등 모든 문자)\W
4. re.compile(‘’) 형식으로 패턴을 반복사용 가능한 객체로 컴파일 한다.  
   이를 a=re.compille(“[^a-z]+”)로 두고 re.findall(a,원하는 문자열) 이런식으로 쓴다.  
   이는 a.findall(‘문자열’)로 쓰면 re를 안쓰고 절약할 수 있다.  
   re.compile(“<title>(.+)</title>”)로 태그 안의 모든 글자를 추출할 수 있다.  
   re.sub(‘원하는 글자’,’그 글자를 덮어쓸 글자’,’문자열’) 문자 치환한 문자열을 반환  
   마찬가지로 a.sub(‘덮어쓸 글자’,’문자열’)로 쓸 수 있다.  
   re.split(‘원하는 글자’,’문자열’) 그 문자를 제외하고 나눈 리스트 반환  
   re.search(‘원하는 글자’,’문자열’) 문자열에서 원하는 문자를 찾아주고 주로  
   re.search().group()으로 원하는 문자를 찾아서 문자열로 뱉어주는 애랑 같이 쓴다
5. 기타  
   문자열 앞에 r’abcd(\n)’을 쓰면 이스케이프 기능을 적용하지 않고 그냥 텍스트를 표기해주는데 주로 파이썬 안에서 정규식과 같은 복잡한 텍스트를 다룰 때 쓴다.

|  |  |
| --- | --- |
|  | |
| 둘 중에 하나로 분류하는 Binary Classification을 알아보자. | |
|  | | 이는 대표적으로 스팸메일 분류, sns feed 노출, 신용카드 기존소비 패턴 인식해 도난 여부 확인 등에 쓰인다. |
|  | | 악성 종양 여부, 주식 구입 여부 등에도 쓰인다. |
|  | | H(x)=Wx+b로 우리가 주로 regression을 쓰는데 이때 이 H(x)값은 0~1 사이가 아니다. |
|  | | 그런데 우리가 내리는 결정은 0 아니면 1이기 때문에 이 사이 값을 받는 것이 판단하기에 편하다. |
|  | | 따라서 0<=g(h(x))<=1 인 g함수를 사람들이 많이 찾으려 하였다. 이 함수는 여러가지가 있는데 |
|  | | 대표적으로 sigmoid함수가 있다. g(z)=1/(1+e^-z)이다. 이를 sigmoid function혹은 logistic function이라 한다. |
|  | | 따라서 최종적으로 H(X)=1/(1+e^-WX)로 가설 함수를 만들었다. |
|  | | 그런데 원래 우리가 썻던 Cost function은 cost(W,b)=1/m\*sigam(H(x)-y)^2의 꼴이었는데 이 함수를 그대로 사용하면 |
|  | | 원래의 매끈한 U자 형이 아니라 매우 구불구불한 U자형이 나오게 된다. |
|  | | 이는 기존 가설 함수가 선형이라 그를 제곱한 꼴인 cost 함수가 2차 모습을 보이는 것과 비교할 때 |
|  | | sigmoid 함수는 S거꾸로 모양을 취하고 있기 때문에 거기에 제곱형태로 하면 구불구불해지는 것이다. |
|  | | 그런데 이렇게 구불구불하면 local minimum이 도출된다. 따라서 global minimum을 도출하지 못하게 되어 새로운 cost 함수가 필요하다. |
|  | |  |
|  | | 따라서 C(H(x),y)=-log(H(x)) :y=1 / -log(1-H(x)) :y=0 을 쓰자. |
|  | | 먼저 y=1 즉, 실제로 1일때 우리의 가설 함수H(x)가 1이라고 추측을 한다면 log(1)=0으로 손실이 없다는 것을 표현 가능하고 |
|  | | 반대로 0을 도출한다면 -log(0)은 +무한대로 손실이 매우 크다고 표현할 수 있다. |
|  | | 이렇게 양쪽 함수를 합치면 우리가 좋아하는 U자형 함수를 도출 할 수 있다. |
|  | | 그런데 이는 if 조건이 있어 코딩시 복잡하기 때문에 |
|  | | C=-ylog(H(x)) - (1-y)log(1-H(x))으로 하나의 함수로 만든다. |
|  | | 이는 우리가 원하는 실제 값y가 0과 1 두가지 밖에 없기 때문에 한 부분이 0이되는 것을 이용한 방법이다. |
|  | | 여기서 마지막으로 우리가 m으로 나누면 최종 cost함수가 완성되고 이 부분까지 식으로 써주고 미분식은 tesnsor magic을 이용하여 풀자. |
|  | |  |
|  | | [tensorflow code] |
|  | | x\_data=[[1,2],[2,3],[3,1]] |
|  | | y\_data=[[0],[0],[1]] |
|  | | x=tf.placeholder(tf.float32,shape=[None,2]) 2pair씩 주기 때문 |
|  | | y=tf.placeholder(tf.float32,shape=[None,1]) |
|  | | W=tf.Variable(tf.randaom\_normal([2,1]),name='weight') |
|  | | b=tf.Varaiable(tf.random\_normal([1]),name='bias') |
|  | | hypo=tf.sigmoid(tf.matmul(X,W)+b)으로 sigmoid 내장 함수를 쓰면 쉽게 가설함수를 만들수 있다. |
|  | | cost=-(-가 붙는 다는 것에 상당히 유의 log함수 밑 부분을 가져왔기 때문)tf.reduce\_mean(y\*tf.log(hypo)+(1-y)\*tf.log(1-hypo)) |
|  | | train=tf.train.GradientDescentOptimizer(0.01).minimize(cost) #원래 optimizer정의하고 또 minimize한 것을 간략화 |
|  | | predicted=tf.cast(hypo>0.5,dtype=tf.float32) float으로 cast하면 1.0 혹은 0.0의 결과를 내뱉는다. |
|  | | accuracy=tf.reduce\_mean(tf.cast(tf.equal(predicted,y),dtype=float32)) #이는 실제 값과 예측값이 같은지를 판단하고 |
|  | | 다시 한번 float으로 변환한다음에 평균을 취해서 정확도를 분석하는 것이다. |
|  | |  |
|  | | with tf.Session() as sess: |
|  | | sess.run(tf.global\_variables\_initalizer()) |
|  | |  |
|  | | feed={x:x\_date, y: y\_data} |
|  | | for step in range(10001): |
|  | | sess.run(train,feed\_dict=feed) |
|  | |  |
|  | | h,c,a=sess.run([hypo,predicted,accuracy],feed\_dict=feed) |
|  | | print("\nhypo",h,-,c,-,a) |
|  | |  |
| 끝까지 작성한 파일이 날라가 간략하게만 다시 적겠습니다....ㅠ |
|  |  | |
|  | softmax classification은 binary classification 처럼 한 개의 상태이냐 아니냐를 판단하는 것이 아닌 | |
|  | 여러개의 상태 중 어떤 것에 속하는지 분류할때 사용됩니다. | |
|  | 이떄 식은 h(x)=xW+b의 기본 가설함수를 따르지만 | |
|  | input x값은 n\*input으로 주어지는 특성들의 개수 y와 h(x)는 n\*구별해야할 라벨 개수 | |
|  | 따라서 W는 특성개수 \*라벨 개수가 되고 이 행렬의 각 열은 가설행렬의 각 라벨을 결정하게 됩니다. | |
|  |  | |
|  | 이번에는 sigmoid가 아닌 softmax function을 사용하여 기본 logits 벡터들을 모두 0~1사이이고 합하여 1이 되는 | |
|  | 벡터로 바꾸도록 합니다. 이는 각 예측값을 합쳤을때 1이되는 확률 값으로 변환해주기 위함입니다. | |
|  | softmax= e^yi/sigma(e^yj) | |
|  | 결국 우리 가설함수 H(x)=softmax(h(x))가 됩니다. | |
|  |  | |
|  | 이제 cost 함수를 알아볼텐데 이번에 알아볼 cost 함수는 cross-entropy cost funciton으로 | |
|  | D(H(x),y)=sigma(yi\*(-log(H(x)))) 입니다. | |
|  | 이때 i는 라벨의 개수만큼입니다. | |
|  | 이해해보자면 라벨이 1으로 표시된 부분 즉, 실제 그 트레이닝 데이터의 정답인 부분의 예측함수가 1로부터 | |
|  | 멀리 떨어진 만큼을 cost로 설정하는 것이고 이는 앞의 logistic cost와 결국 같은 메커니즘입니다. | |
|  | 마지막으로 이를 전체 트레이닝 데이터의 개수만큼 모두 더하고 나누어 평균을 내면 | |
|  | 우리가 원하는 최종 Loss=1/n\*sigam(D(H(x),y))가 완성됩니다. | |
|  |  | |
|  | [tensorflow code] | |
|  | #기존 logistic 부분과 다른 부분 위주로 설명하겠습니다. | |
|  | #주의할 부분은 이제 y의 데이터 형식이 n\*label이므로 | |
|  | y=tf.placeholder("float",[None,3])이런식으로 써주어야하고 | |
|  | W=tf.Variable(tf.random\_normal([input,label]),name='weight')로 적어야합니다. | |
|  | #이때 가설은 최종적으로 softmax한 것을 가설로 정하므로 | |
|  | hypo=tf.nn.softmax(tf.matmul(x,W)+b) | |
|  | cost=tf.reduce\_mean(-tf.reduce\_sum(y\*tf.log(hypo),axis=1)) | |
|  | #이부분이 조금 이해하기 까다로울 수 있는데 axis=1은 행 안에서 합치라는 뜻입니다. | |
|  | #따라서 실제 y\*H(x)의 모양이 [ [0 0.8 0], [0 0 0.7] ] 처럼 나오고 (elementwise곱이기 떄문) | |
|  | #이를 [0.8, 0.7] 이런식으로 만들어줍니다 | |
|  | #즉 우리의 cross entropy cost function에서 라벨이 아닌 부분은 0이 곱해져 사라졋던 것을 행해주는 부분입니다. | |
|  | #그리고 이를 axis없이 트레이닝 데이터 개수로 평균내면 마지막 스칼라로 된 loss를 얻을 수 있습니다. | |
|  |  | |
|  | #Fancy sofrmax | |
|  | #조금 더 세세한 부분을 다루어 보겠습니다. | |
|  | a=sess.run(hypo,feed\_dict=feed) | |
|  | print(a,sess.run(tf.argmax(a,1))) | |
|  | #이는 행속에서 가장 높은 애의 index를 뱉어내줍니다. | |
|  | cost=tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=logits,labels=y) | |
|  | #이 최신 버젼을 이용하면 tf.matmul(x,W)+b를 만들어 y와 넣으면 바로 스칼라 cost를 구해줍니다. | |
|  | #단 이때 y는 역시 one-hot encodig된 데이터여야겠죠! | |
|  | #만일 그러지 않은 경우 0~6까지 정수형을 되있을 경우에 one-hot화 하는 방법을 알아봅시다. | |
|  | y\_before=tf.placeholder(tf.int32,[None,1]) # 0~6의 자료형이 주어졌을때 | |
|  | y=tf.one\_hot(y,label=7) | |
|  | y=tf.reshape(y,[-1,label=7]) | |
|  | #그냥 온핫하면 [ [[100]] [[011] ] 처럼 차원을 하나 더 주기 때문에 reshape을 이용하여 n\*7의 형태로 바꾸어줍시다. | |
|  |  | |
|  | #나머지는 모두 같고 예측치와 정확도를 한번 구해봅시다. | |
|  | prediction=tf.argmax(hypo,1) #가장 높은 확률이 나온 애로 predict하자고 결정 | |
|  | correct\_prediction=tf.equal(prediction,tf.argmax(y,1))#실제 라벨값과 예측 값이 같으면 1 아니면 0의 행렬화 | |
|  | accuracy=tf.reduce\_mean(tf.cast(correct\_prediction,tf.float32))#다시 float으로 바꾸고 평균을 구한다. | |

LSTMs, convolutions, custom objective functions, multiple cortical columns