영어음성학필기

2015410098 컴퓨터학과문승연

음성 분석 프로그램 Praat와 Python(Jupyter)를 이용해서 직접 음성분석을 할 예정.

음성학이란?
★ A study on speech 말소리 자체에 관한 연구

크게 3가지로 분류할 수 있는데

1. 어떻게 소리가 나오는가? (from mouth)
2. 어떻게 소리가 전달되는가? (through air)-> 공기를 타고 가는 과정
3. 어떻게 소리가 들리는가? (to ear)

이중에서 1번이 음성학의 가장 주된 부분이다.

Articulation (소리를 입에서 만들어내는 과정)
Vocal tract는 코 쪽 통로인 nasal tract와 입 쪽 통로인 oral tract로 나눌 수 있다. 그리고 목 안쪽의 larynx라는 부위가 존재한다.
먼저 구강통로의 위 쪽 부분은 lip, teeth, alveolar, hard palate, soft palate(velum), uvula, larynx가존재한다 (순서대로 입술, 이, 치경, 경구개, 연구개, 목젖, 후두)
반대로 아래 부분은 lip, tip, blade, front, center, back, root, epiglottis가 존재한다 (순서대로 입술, 혀끝, 혓날, 혀의 앞부분, 몸통, 뒷부분, 혀뿌리, 후두 덮개)

또 연구개(velum)는 비음이 아닌 소리를 낼 때 비강으로 가는 통로를 막는 역할을 한다. 코로 숨을쉴 때는 비강통로가 열려야하므로 연구개가 내려가고 비음이 아닌 소리를 낼 경우에는 연구개가 올라가서 비강통로를 막는 구조이다. 비음의 예시로는 m, n, ng 등의 소리가 해당된다. 이 소리들은 비강통로를 통해서 소리가 나므로 소리를 내는 도중 코를 막으면 제대로 발음할 수 없다. 또한 우리가 일상생활에서 숨을 쉴 때 코를 통해서 숨을 쉬므로 평상시에는 연구개가 내려와서 비강통로가 열려있는 상태라는 것을 어렵지 않게 유추할 수 있다.

목 안쪽에는 voice box라고도 불리는 후두(larynx)가 존재하는데 소리의 종류가 유성음이냐 무성음이냐에 따라서 후두가 진동하거나 진동하지 않는다. V,z, l, m 등의 유성음을 낼 경우 후두가 진동하는 것을 느낄 수 있다.

소리를 낼 때 각 constrictor가 사용된다 하더라도 어디에 닿을 것인지? 얼마나 막을 것인지? 이러한 요소들에 따라서 소리가 달라진다.

Constriction Location(CL)의 차이에 의한 소리 변화의 예시로, 아랫입술이 윗입술에 닿느냐 또는 윗니에 닿느냐에 따라 소리가 달라지고 마찬가지로 혀끝이 이빨에 닿느냐 치경에 닿느냐 경구개에 닿는냐에 따라 소리가 변한다.

Constriction Degree(CD)의 예시로 파열음, 마찰음, 파찰음 등의 차이를 들 수 있다.

소리를 포함한 이 세상의 모든 신호들은 단순한 형태의 sine wave가 여러개 합쳐져서 만든 형태이다. 따라서 아무리 복잡한 소리라도 분석해보면 여러가지 frequency의 sine wave들의 합으로 표현할 수 있다. praat를 이용해서 이를 직접 눈으로 확인해 볼 수 있다.

먼저 스스로의 목소리를 녹음한 다음 그 포만트를 분석해보자. 일정한 주기로 파동이 반복되고 있음을 관찰할 수 있는데 그 주기를 주파수로 바꾸면 바로 그 주파수가 자신의 목소리의 주파수임을 알 수 있다. 필자의 경우 딱 100Hz로 나타났다.

그 다음 자신의 목소리의 주파수와 같은 주파수를 가지는 pure tone을 만든다음 그 소리를 비교해보자. 기계음이기 때문에 사람의 목소리와는 다르지만 같은 음을 내는 것을 알 수 있다. 이때 목소리가 다르게 느껴지는 이유는 그 소리의 source와 소리를 내는 과정에서의 filter의 차이가 있기 때문이다. 여기서 filter란 목소리를 내는 과정에서 발생하는 변형, 즉 입모양으로 인한 변형을 뜻한다.

말소리의 source는 larynx의 소리이다. Siswati어를 하나는 직접 말소리를 녹음하고 또 다른 하나는 사람의 목, larynx부분에 대고 녹음한 소리를 비교해보면 톤이나 음의 높이등은 같지만 발음에서 차이가 느껴지는 것을 알 수 있다. 이는 larynx의 소리는 입모양으로 인해 filtering 되지 않은 소리이기 때문이다.

사람 목소리처럼 여러개의 pure tone(sine wave)의 합으로 이루어진 tone을 complex tone이라고 한다. 사람의 목소리의 주파수가 100Hz라고 가정하면 그 주파수가 Fundamental frequency가 되고 이 배수만큼의 주파수들이 합해져서 complex tone이 만들어진다.

예를 들자면 100Hz가 Fundamental frequency(F0)이고 그 다음 2배수인 200Hz의 sine wave가 F0의 크기보다 낮은 크기로 더해지고 그다음 300Hz의 sine wave도 더해지고... 이런 식으로 계속 반복되어 complex tone이 만들어진다.

실제로 praat를 이용해서 녹음한 목소리의 spectrum을 분석해보면 파동의 peak부분이 자신의 f0인 주파수의 배수마다 나타나는 것을 확인할 수 있다.

이는 실제로 100Hz부터 1000Hz까지의 pure tone을 amplitude를 0.05씩 낮춰가면서 합성해 complex tone을 만든 뒤 그 spectrum을 분석해봐도 확인할 수 있다.

이때 praat상에서 spectrum분석을 통해 볼 수 있는 spectrum을 위에서 본 모양으로 진한 부분과 연한 부분으로 나타낸 그래프를 spectrogram이라고 한다. 진한 부분이 peak에 해당하는 부분으로 formants라고도 한다.
일반적으로 모음을 비교할 때에는 Formant 1과 Formant 2를 비교해보면 충분하다. 이를 바탕으로 해서 F1과 F2의 위치에 따라 날 수 있는 모음의 종류를 나타낸 그래프를 vowel space라고 한다.

- Python

- Variables(변수)

파이썬에는 여러가지 형태의 변수들이 존재한다.

1. 정수형 데이터를 나타내는 int
2. 소수형 데이터를 나타내는 float
3. 문자열 데이터를 나타내는 string(str)
4. 여러가지 데이터를 한꺼번에 담을 수 있는 리스트형태의 데이터를 나타내는 list
5. List와 마찬가지로 여러가지 형태의 데이터를 담을 수 있지만 그 내용을 변경할 수 없는 tuple
6. Key, value의 쌍으로 이루어진 데이터들을 담는 dictionary (dict)

   등을 예시로 들 수 있다.

Import numpy as np 명령어를 실행하면 numpy 라이브러리를 import할 수 있다.

Numpy는 행렬 계산을 할 때 매우 유용하다.

b = np.array(a)라는 명령어로 변수 b를 선언하면 b는 numpy 타입을 갖는 하나의 변수가 된다.

numpy는 기본적인 행렬의 사칙연산을 지원한다.

matrix 함수를 이용하면 하나의 행렬을 만들 수 있는데 이를 이용해서 행렬의 합차는 물론 곱연상도 가능하다.

- String (문자열)

s = "abcdef" # 이때 s는 문자열 값을 값는 변수이다.

파이썬에서는 문자열을 이루는 각 글자마자 인덱스를 가지고 있고 이를 이용해서 특정 순서의 글자만을 추출할 수 있다.

"abcdef"에서 각 글자 "a", "b", "c", "d", "e", "f"는 순서대로 0, 1, 2, 3, 4, 5 인덱스를 가진다.

따라서 b[0], b[1], b[2], b[3], b[4], b[5]를 입력하면 순서대로 "a", "b", "c", "d", "e", "f"를 리턴한다.

이는 list 타입 변수에서도 비슷하게 적용할 수 있다.

Ex) n = [100, 200, 300]

n[0], n[1], n[2] 는 순서대로 100, 200, 300을 리턴한다.

추가로 응용하여 n[0:], n[1:2] 처럼 범위를 지정해서 값을 불러올 수도 있다. 이 경우 리스트는 리스트 형태로 그 범위에 해당하는 값을 리턴한다. (문자열은 문자열을 리턴한다.)

또한 문자열 형태의 변수에는 여러가지 메소드가 존재한다.

1. upper() : 소문자 문자를 대문자로 변경해서 리턴
2. find(str) : 특정 문자열을 찾아서 해당 문자열이 시작하는 첫번째 글자의 인덱스를 리턴 (없을 경우 -1리턴)
3. rindex(str) : 특정 문자열를 찾아서 인덱스를 리턴하되 가장 뒤에 존재하는 문자열의 인덱스를 리턴 (없을 경우 ValueError 발생)
4. strip() : 문자열의 앞 뒤의 공백 등의 whitespace 제거한 후 리턴
5. split(str) : 문자열을 특정 문자열을 기준으로 나누어 리스트 형태로 리턴 (default 공백)
6. join(tokens) : 리스트에 존재하는 문자열들 사이사이에 특정 문자열을 삽입
7. replace(str1, str2) : 첫번째 문자열을 두번째 문자열로 변경한 후 리턴

- syntax (문법)

for 구문

for문은 기본적인 반복문으로서 파이썬에서는 리스트나 문자열처럼 indexable한 변수들을 이용해서 for문을 만들 수 있다.

for 구문을 만들 수 있는 방법으로는 다음과 같다.

1. 리스트

2. 문자열
3. range 함수를 이용해 직접 범위 설정
4. enumerate함수를 이용해 인덱스와 indexable 변수의 값 둘 다 이용
5. zip함수. 4번과 유사함

if 구문은 조건문으로써 특정 조건을 걸고 조건에 해당하는 경우에만 조건문 내의 명령을 실행하도록 하는 구문이다.

```
a = 0
if a == 0:
        print(a) # a = 0 이므로 이 구문이 실행됨.
else:
        print(a+1) # a가 0이 아니라면 이 구문이 실행된다.
```

또한 list를 선언할 때 for문을 이용해서 내용물을 채우는 것이 가능하다.

```
Ex) a = [1,2,3,4]
 b = [j for j in a if j >= 2]
```

a의 있는 값들 중 2보다 크거나 같은 값들로만 순서대로 list가 만들어진다.

따라서 b는 [2, 3, 4]가 된다.

Numpy Tutorial (10월 마지막 주)

Import numpy as np (as는 모듈명이 기니까 줄여서 쓰기 위함.)

Import matplotlib.pyplot as plt (matplotlib의 pyplot모듈을 plt라는 이름으로 쓰겠다는 뜻)

```
np.empty([2,3], dtype='int')
```

➔ int 타입 변수를 값으로 가지는 2행 3열짜리 ndarray 객체를 만듬

np.zeros

➔ 모든 값이 0인 ndarray 객체를 만듬

np.arange(0, 10, 2, dtype="float64")

➔ 0에서 10까지 2단위로 float64형태의 값을 가지는 ndarray 객체 생성

np.linspace(0,10,6, dtype=float)

➔ 0에서 10까지 값을 6등분하여 ndarray 객체 생성

X = np.array([[1,2,3],[4,5,6]])

➔ 첫번째 행에는 [1,2,3] 두번째 행은 [4,5,6]을 값으로 가지는 2행 3열 행렬

X.astype(np.float64)

➔ 타입을 int에서 float64로 바꿈

np.zeros_like(X)

➔ X와 같은 행, 열을 가지고 모든 값이 0인 행렬을 리턴함

```
data = np.random.normal(0,1,100)
# data는 중앙값이 0  표준편차가 1 크기(변수의 개수)가 100이고 정규분포를 따르는
# 데이터들의 집합(행렬)
plt.hist(data, bins=10)
plt.show() # 위의 data를 10개의 바(bar)를 가지는 히스토그램으로 표현.

Y = X.reshape(-1, 3, 2)
# X 행렬을 값은 그대로 하고 행과 열의 개수 등만 바꿔서 표현, 이때 -1은 디폴트
# 여기서는 4에 해당한다

np.allclose(X.reshape(-1,3,2), Y) # 두 행렬이 서로 같은지 비교. 값은 True or False

assert np.allclose(X.reshape(-1,3,2), Y) # assert는 가정설정문으로써 assert 다음에 나오는 구문이
```

False일 경우 AssertionError를 발생시킨다.

a = np.random.randint(0,10,[2,3]) # 0에서 10 사이의 int변수를 2행 3열 행렬에 담음

b = np.random.random([2,3]) # 랜덤 변수로 2행 3열 행렬을 만듬

np.savez("test", a,b) # test파일에 a와 b를 저장 (npz파일)

npzfiles = np.load("test.npz") # npz파일 로드

data = np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("X", "Y"), 'formats':('f', 'f')})

"regression.csv"라는 파일을 load 각각 (X값, Y값) 의 형태로 float포맷을 갖도록 함.

np.savetxt("regression_saved.csv", data, delimiter=",") # csv파일 저장

a = np.arange(1,5)

b = np.arange(9,5,-1)

print(a-b)

print(a*b) # 각각의 numpy.ndarray의 인덱스에 맞는 값들을 빼거나 곱함

a = np.matrix(a)

b = np.matrix(b) # a와 b를 행렬 연산을 하기 위해 ndarray가 아닌 matrix객체로 바꿔줌

matrix객체를 서로 곱하면 각 인덱스 간의 곱이 아닌 행렬의 곱셈과 같이 연산을 함.

a = np.arange(1,10).reshape(3,3)

b = np.arange(9,0,-1).reshape(3,3)

a == b # 각 ndarray객체의 값들 별로 비교

a.sum(), np.sum(a) # 둘이 같은 값 # 각 행렬의 값들의 총 합을 리턴

a.sum(axis=0), np.sum(a,axis=0) # 각 열 별로 연산

a.sum(axis=1), np.sum(a, axis=1) # 각 행 별로 연산

a = np.arange(1,25).reshape(4,6)

b = np.arange(6)

c = np.arange(4).reshape([4,1])

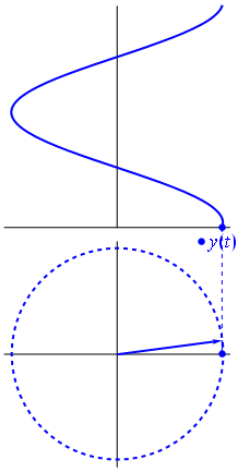
a+b를 하면 a의 각 행의 값들마다 그 인덱스에 맞는 b의 값을 더함

a+c도 마찬가지로 a의 각 열의 값들마다 그 인덱스에 맞는 c의 값을 더함

a+100과 같이 상수를 더할 경우 a의 모든 값에 그 상수만큼 더함.

Phasor란?

페이저(phasor)는 오일러 공식 $e^{i\theta} = \cos\theta + i\sin\theta$ 을 이용해 시간에 대해 진폭, 위상, 주기가 불변인 함수를 표현하는 방법이다. 대표적인 예시로 sin, cos그래프가 있다. 페이저를 이용하면 복소평면상의 실수값 cos(theta)와 허수값 sin(theta)를 복잡한 삼각함수 연산이 아닌 복소수의 계산으로 대체할 수 있다.



(phasor 그래프를 나타낸 예시)

시간이 지남에 따라 일정 주파수로 theta가 변하면 cos(theta) + isin(theta)값도 일정 주기를 갖고 반복하는 그래프를 그린다. 이번 시간에는 그 모습을 파이썬을 이용해 직접 plot 해보도록 한다.

주기(dur)를 0.5s, 샘플 주파수(sr)를 10,000Hz, sine 진동수(freq)를 100Hz라고 하자.

이때 시간 t를 1/10000초부터 5000/10000초까지 균등하게 분할한다. (일정 시간에 따른 위상 변화를 관측하기 위해)

t = np.arange(1, sr * dur + 1)/sr

➔ t = array([1.000e-04, 2.000e-04, 3.000e-04, … , 4.999e-01, 5.000e-01])

각 theta는 주파수 freq만큼 반복시킬 것이기 때문에 t * 2pi에다가 freq만큼 곱한 값이 된다.

theta = t * 2*np.pi * freq

➔ theta = array([6.28318531e-02, 1.25663706e-01, 1.88495559e-01, ..., 3.14033602e+02, 3.14096434e+02, 3.14159265e+02])

이제 코사인 페이저 signal을 generate한다.

s = np.sin(theta)

fig = plt.figure()

ax = fig.add_subplot(111) # 여기서 add_subplot함수는 인자 값에 따라 plot을 몇 개 그릴지 그리고 그 중 몇번째 위치에 그릴지 정한다. 111의 경우 1X1 크기에서 1번째 자리에 그린다는 뜻.

ax.plot(t[0:1000], s[0:1000], '.')

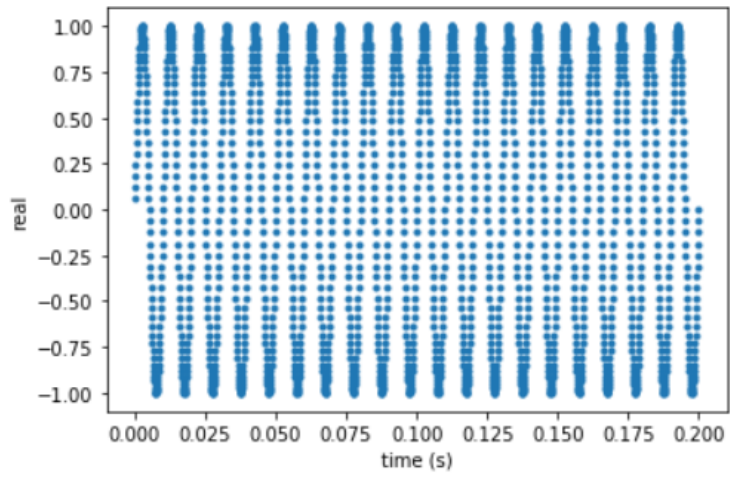
범위를 0부터 10000까지 하면 그래프 간격이 너무 좁아 알아보기 힘드므로 0부터 1000까지만 범위로 지정해준다.

ax.set_label('time(s)')

ax.set_label('real')

이렇게 하면 코딩 결과는 아래와 같다.

```
Text(0, 0.5, 'real')
```



여기서 sin함수가 아닌 자연상수를 이용한 complex-phasor를 구현할 수도 있다.

c = np.exp(theta*1j) # c = e^i*theta와 같다. 그리고 이는 위에서 설명한 바와 같이 cos(theta) + i*sin(theta)의 값과 같다.

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d') # 3차원 그래프

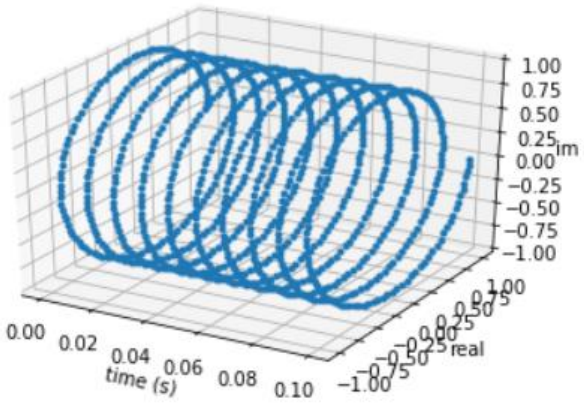
ax.plot(t[0:1000], c.real[0:1000], c.imag[0:1000]. '.')

ax.set_xlabel('time(s)')

ax.set_ylabel('real')

ax.set_zlabel('imag')

```
Text(0.5, 0, 'imag')
```



iPython 모듈 중에서도 display 모듈을 이용하면 페이저 함수를 소리의 형태로 재생하는 것이 가

능하다.

ipd.Audio(s,rate=sr)

위의 sound는 하나의 주파수 값만을 가지는 소리에 불과하다. 하지만 실제 사람의 목소리는 하나의 주파수가 아닌 fundamental frequency(f0)와 그 배수 주파수 값의 합성으로 이루어져 있다.

praat에서 했던 각 주파수에 해당하는 pure tone을 합치는 과정을 python 코딩으로도 할 수 있다.

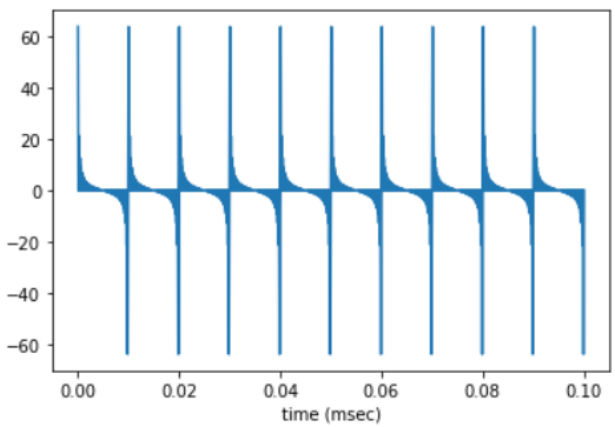
이때 Nyquist Frequency(나이퀴스트 주파수)라는 개념을 알아야 하는데.

먼저 나이퀴스트 정리를 간략하게 설명하자면, 샘플링 레이트(sampling rate)가 표현하고자 하는 신호 주파수의 2배 이상이 되어야 완벽하게 표현할 수 있다는 뜻이다. 샘플링 레이트보다 큰 주파수의 신호가 들어오면 주어진 샘플링 레이트로 모든 점을 완벽하게 복원하는 것이 불가능하기 때문이다.

이를 바탕으로 코드를 짜면 아래와 같은 결과가 나온다.

```
F0 = 100; Fend = int(sr/2); s = np.zeros(len(t));
for freq in range(F0, Fend+1, F0):
    theta = t * 2*np.pi * freq
    s = s + amp * np.sin(theta)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(t[0:1000], s[0:1000]);
ax.set_xlabel('time (msec)')
ipd.Audio(s, rate=sr)
```

Out[14]:



하지만 위 그래프는 실제 음성과 달리 주파수가 커져도 그 크기(진폭)이 감소하지 않고 그대로인 상태로 합성시킨 결과이다. 즉, 위 그래프는 스펙트럼이라고 할 수 없다.

따라서 주파수가 증가할수록 진폭을 완만하게 감소시키는 함수를 하나 만든다.

```
def resonance (srate, F, BW): # 완만하게 내려가는 산을 만드는 작업
    a2 = np.exp(-hz2w(BW,srate))
    omega = F*2*np.pi/srate
    a1 = -2*np.sqrt(a2)*np.cos(omega)
    a = np.array([1, a1, a2])
    b = np.array([sum(a)])
    return a, b
```

그리고 이 함수를 이용해서 여러 주파수의 sine wave를 합성시킬수록 실제 음성과 가까운 소리를 만들어낼 수 있다.