Speed Dating Analysis project

Contents

프로젝트 소개

Speed Dating

남녀 소개팅 매칭 요소 분석 및 예측 프로그램 만들기

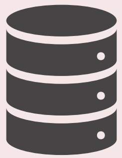
Speed dating의 밤에서 남녀가 소개팅을 한 후 설문에 응답한 결과를 바탕으로 어떤 남녀가 매칭될 확률이 높을지 예측하는 것을 목표로 한다.

Speed dating

사람 수가 많은 그룹에서 아이스 브레이킹이나 커뮤니케이션 방식으로 활용할 수 있는 간단한 테크닉이다. 말 그대로 데이트를 하는 것 같이, 아주 제한된 시간 동안 2명씩 짝을 지어서 특정한 주제에 대해 서로 번갈아 얘기하게 한다. 이 방법은 그룹 멤버들 간의 친밀감을 높여주며, 모든 사람들이 서로의 목소리를 들을 수 있게 해준다.

데이터 소개

데이터 소개



Speed dating의 밤에서 남녀가
소개팅을 한 후 설문에 응답한 결과

출처 : OpenML SpeedDating

123 features

Speed
Dating

데이터 소개

123 features

- **gender** : Gender of self
- **age** : Age of self
- **age_o** : Age of partner
- **d_age** : Difference in age
- **race** : Race of self
- **race_o** : Race of partner
- **samerace** : Whether the two persons have the same race or not.
- **importance_same_race** : How important is it that partner is of same race?
- **importance_same_religion** : How important is it that partner has same religion?
- **field** : Field of study

자신과 상대방의 기본정보

자신의 성별
자신과 상대방의 나이
상대방과의 나이차이

나와 상대방의 인종
상대방과 인종이 같은 지 여부
인종과 종교의 중요도

전공분야

데이터 소개

- **pref_o_attractive** : How important does partner rate attractiveness
- **pref_o_sinsere** : How important does partner rate sincerity
- **pref_o_intelligence** : How important does partner rate intelligence
- **pref_o_funny** : How important does partner rate being funny
- **pref_o_ambitious** : How important does partner rate ambition
- **pref_o_shared_interests** : How important does partner rate having shared interests
- **attractive_o** : Rating by partner (about me) at night of event on attractiveness
- **sincere_o** : Rating by partner (about me) at night of event on sincerity
- **intelligence_o** : Rating by partner (about me) at night of event on intelligence
- **funny_o** : Rating by partner (about me) at night of event on being funny
- **ambitous_o** : Rating by partner (about me) at night of event on being ambitious
- **shared_interests_o** : Rating by partner (about me) at night of event on shared interest

- **attractive_important** : What do you look for in a partner - attractiveness
- **sincere_important** : What do you look for in a partner - sincerity
- **intellicence_important** : What do you look for in a partner - intelligence
- **funny_important** : What do you look for in a partner - being funny
- **ambtition_important** : What do you look for in a partner - ambition
- **shared_interests_important** : What do you look for in a partner - shared interests
- **attractive** : Rate yourself - attractiveness
- **sincere** : Rate yourself - sincerity
- **intelligence** : Rate yourself - intelligence
- **funny** : Rate yourself - being funny
- **ambition** : Rate yourself - ambition
- **attractive_partner** : Rate your partner - attractiveness
- **sincere_partner** : Rate your partner - sincerity
- **intelligence_partner** : Rate your partner - intelligence
- **funny_partner** : Rate your partner - being funny
- **ambition_partner** : Rate your partner - ambition
- **shared_interests_partner** : Rate your partner - shared interests

상대 평가 항목

매력 / 성실성 / 지능 / 유머 / 의욕 / 자산

위 항목을 기준으로

- 중요도
- 상대방의 나에 대한 평가
- 파트너에 대해 어떻게 느꼈는지
- 스스로를 평가
- 파트너를 평가

데이터 소개

- **sports** : Your own interests [1-10]
- **tvsports**
- **exercise**
- **dining**
- **museums**
- **art**
- **hiking**
- **gaming**
- **clubbing**
- **reading**
- **tv**
- **theater**
- **movies**
- **concerts**
- **music**
- **shopping**
- **yoga**
- **interests_correlate** : Correlation between participant's and partner's ratings of interests.

참가자의 취미

- 17개의 취미에 대한 흥미 정도
- 참가자와 파트너의 취미 상관 관계

데이터 소개

- **expected_happy_with_sd_people** : How happy do you expect to be with the people you meet during the speed-dating event?
- **expected_num_interested_in_me** : Out of the 20 people you will meet, how many do you expect will be interested in dating you?
- **expected_num_matches** : How many matches do you expect to get?
- **like** : Did you like your partner?
- **guess_prob_liked** : How likely do you think it is that your partner likes you?
- **met** : Have you met your partner before?
- **decision** : Decision at night of event.
- **decision_o** : Decision of partner at night of event.
- **match** : Match (yes/no)

Speed dating event에 대한 기대와 결과

- Speed-dating event에서 만난 사람과 잘 될 것이라고 예상하는지
- 나에게 관심을 보일 것으로 예상되는 사람의 수
- 매칭되기 예상되는 사람의 수
- 파트너가 맘에 들었는지
- 얼마나 맘에 들었는지
- 이벤트의 밤에 결정
- 상대방의 결정
- 매칭 여부 (Yes or No)

데이터 전처리 과정

3

003 데이터 전처리 과정

```python
import csv
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.linear_model import SGDClassifier
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn import neighbors, datasets, linear_model
from sklearn import decomposition
from sklearn import ensemble
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc, accuracy_score
from sklearn import cluster
# from sklearn.metrics import accuracy_score
```

```python
# GradientBoosting용
from sklearn import datasets
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error

# svm용
from sklearn import datasets, model_selection, metrics, svm
from sklearn.externals import joblib
from sklearn.externals import joblib

#knn
from sklearn import neighbors, datasets
from sklearn.metrics import accuracy_score


df_data = pd.read_csv('speeddating.csv', na_values = '?', encoding='utf-8') # 엑셀 파일 읽기
df_data.head()
```

003 데이터 전처리 과정

```
y_data = df_data[['match']]
y_data.head()
```

	match
0	0
1	0
2	1
3	1
4	1

Y_data : ‘match’

‘match’ : Speed dating event 후 매칭 성공 여부

```
x_data = df_data.copy()
del x_data['match']

x_data.head()
```

	has_null	wave	gender	age	age_o	d_age	d_d_age	race	race_o	samerace	importance_same_race	importance_same_religion	d_
0	0	1	female	21.0	27.0	6	[4-6]	'Asian/Pacific Islander/Asian-American'	European/Caucasian-American	0	2.0	4.0	
1	0	1	female	21.0	22.0	1	[0-1]	'Asian/Pacific Islander/Asian-American'	European/Caucasian-American	0	2.0	4.0	
2	1	1	female	21.0	22.0	1	[0-1]	'Asian/Pacific Islander/Asian-American'	'Asian/Pacific Islander/Asian-American'	1	2.0	4.0	
3	0	1	female	21.0	23.0	2	[2-3]	'Asian/Pacific Islander/Asian-American'	European/Caucasian-American	0	2.0	4.0	
4	0	1	female	21.0	24.0	3	[2-3]	'Asian/Pacific Islander/Asian-American'	'Latino/Hispanic American'	0	2.0	4.0	

003 데이터 전처리 과정

결측치 확인

```
# 결측치 확인
x_data.isna().sum() # x_data.isnull().sum()(
```

```
has_null                          0
wave                              0
gender                            0
age                              95
age_o                           104
d_age                             0
d_d_age                           0
race                             63
race_o                           73
samerace                          0
importance_same_race             79
importance_same_religion         79
d_importance_same_race            0
d_importance_same_religion        0
field                            63
pref_o_attractive                89
pref_o_sincere                   89
pref_o_intelligence              89
pref_o_funny                     98
```

```
pref_o_shared_interests         129
d_pref_o_attractive               0
d_pref_o_sincere                  0
d_pref_o_intelligence             0
d_pref_o_funny                    0
d_pref_o_ambitious                0
d_pref_o_shared_interests         0
attractive_o                    212
sinsere_o                       287
intelligence_o                  306
                               ...
d_exercise                        0
d_dining                          0
d_museums                         0
d_art                             0
d_hiking                          0
d_gaming                          0
d_clubbing                        0
d_reading                         0
d_tv                              0
```

```
d_concerts                        0
d_music                           0
d_shopping                        0
d_yoga                            0
interests_correlate             158
d_interests_correlate             0
expected_happy_with_sd_people   101
expected_num_interested_in_me  6578
expected_num_matches           1173
d_expected_happy_with_sd_people   0
d_expected_num_interested_in_me   0
d_expected_num_matches            0
like                            240
guess_prob_liked                309
d_like                            0
d_guess_prob_liked                0
met                             375
decision                          0
decision_o                        0
Length: 122, dtype: int64
```

데이터 전처리

```
# 결측치 상위 두 항목 열 지우기
del x_data['expected_num_interested_in_me']
del x_data['expected_num_matches']

# 매칭 예측 시 필요없다고 생각되는 열 제거
del x_data['d_expected_num_interested_in_me']
del x_data['d_expected_num_matches']
```

```
x_data['gender'] = x_data['gender'].replace(['male', 'female'], [0,1]) # 텍스트 숫자로 교체
x_data.head()
```

데이터 전처리

1) nnnew_df -> class_df

범주형으로 변환한 전처리 데이터

2) nnnnew_df -> num_df

수치형을 유지한 전처리 데이터

데이터 전처리

nnew_df 실수 범위의 열들을 구간 범위로 전환

```
nnew_df = x_data.copy()
```

```
nnew_df['d_d_age']=nnew_df['d_d_age'].replace(['[0-1]', '[2-3]', '[4-6]', '[7-37]'], [0,1,2,3])
nnew_df['d_importance_same_race']=nnew_df['d_importance_same_race'].replace(['[0-1]', '[2-5]', '[6-10]'], [0,1,2])
nnew_df['d_importance_same_religion']=nnew_df['d_importance_same_religion'].replace(['[0-1]', '[2-5]', '[6-10]'], [0,1,2])
nnew_df['d_pref_o_attractive']=nnew_df['d_pref_o_attractive'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_pref_o_sincere']=nnew_df['d_pref_o_sincere'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_pref_o_intelligence']=nnew_df['d_pref_o_intelligence'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_pref_o_funny']=nnew_df['d_pref_o_funny'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_pref_o_ambitious']=nnew_df['d_pref_o_ambitious'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_pref_o_shared_interests']=nnew_df['d_pref_o_shared_interests'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_attractive_o']=nnew_df['d_attractive_o'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_sinsere_o']=nnew_df['d_sinsere_o'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_intelligence_o']=nnew_df['d_intelligence_o'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_funny_o']=nnew_df['d_funny_o'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_ambitous_o']=nnew_df['d_ambitous_o'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_shared_interests_o']=nnew_df['d_shared_interests_o'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_attractive_important']=nnew_df['d_attractive_important'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_sincere_important']=nnew_df['d_sincere_important'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_intellicence_important']=nnew_df['d_intellicence_important'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_funny_important']=nnew_df['d_funny_important'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_ambtition_important']=nnew_df['d_ambtition_important'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_shared_interests_important']=nnew_df['d_shared_interests_important'].replace(['[0-15]', '[16-20]', '[21-100]'], [0,1,2])
nnew_df['d_attractive']=nnew_df['d_attractive'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_sincere']=nnew_df['d_sincere'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_intelligence']=nnew_df['d_intelligence'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_funny']=nnew_df['d_funny'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_ambition']=nnew_df['d_ambition'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_attractive_partner']=nnew_df['d_attractive_partner'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_sincere_partner']=nnew_df['d_sincere_partner'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_intelligence_partner']=nnew_df['d_intelligence_partner'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_funny_partner']=nnew_df['d_funny_partner'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_ambition_partner']=nnew_df['d_ambition_partner'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_shared_interests_partner']=nnew_df['d_shared_interests_partner'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_expected_happy_with_sd_people']=nnew_df['d_expected_happy_with_sd_people'].replace(['[0-4]', '[5-6]', '[7-10]'], [0,1,2])
nnew_df['d_like']=nnew_df['d_like'].replace(['[0-5]', '[6-8]', '[9-10]'], [0,1,2])
nnew_df['d_guess_prob_liked']=nnew_df['d_guess_prob_liked'].replace(['[0-4]', '[5-6]', '[7-10]'], [0,1,2])
```

데이터 전처리

class_df: nnew_df에서 범주형 feature 형식 제외하고 삭제

```
nnnew_df = nnew_df.copy()
```

```
del nnnew_df ['wave']
del nnnew_df ['d_interests_correlate']
del nnnew_df ['importance_same_race']
del nnnew_df ['importance_same_religion']
del nnnew_df ['decision']
del nnnew_df ['decision_o']
del nnnew_df ['expected_happy_with_sd_people']
del nnnew_df ['has_null']
```

```
for d in nnnew_df.loc[:,'age':'d_age']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'race':'race_o']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'pref_o_attractive':'pref_o_shared_interests']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'attractive_o':'shared_interests_o']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'attractive_important':'shared_interests_important']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'attractive':'d_ambition']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'attractive_partner':'shared_interests_partner']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'sports':'d_yoga']:
    del nnnew_df [d]

for d in nnnew_df.loc[:,'like':'guess_prob_liked']:
    del nnnew_df [d]

# 범주형 feature 형식 제외하고 삭제
```

3

003 데이터 전처리 과정

데이터 전처리_1) class_df

class_df

실수 범위의 열들을 구간 범위로 전환 → 범주형 feature 형식 제외하고 삭제 → 범주형으로 변환한 전처리 데이터

```
nnnew_df.head() # 범주형으로 변환한 전처리 데이터
```

	gender	d_d_age	samerace	d_importance_same_race	d_importance_same_religion	field	d_pref_o_attractive	d_pref_o_sincere	d_pref_o_intelligence	d_p
0	1	2	0	1	1	Law	2	1	1	
1	1	0	0	1	1	Law	2	0	0	
2	1	0	1	1	1	Law	1	1	1	
3	1	1	0	1	1	Law	2	0	0	
4	1	1	0	1	1	Law	2	0	1	

3

003 데이터 전처리 과정

데이터 전처리_1) class_df

```
nnnew_df.isna().sum() # 결측치 확인
```

```
gender                            0
d_d_age                           0
samerace                          0
d_importance_same_race            0
d_importance_same_religion        0
field                            63
d_pref_o_attractive               0
d_pref_o_sincere                  0
d_pref_o_intelligence             0
d_pref_o_funny                    0
d_pref_o_ambitious                0
d_pref_o_shared_interests         0
d_attractive_o                    0
d_sinsere_o                       0
d_intelligence_o                  0
d_funny_o                         0
d_ambitous_o                      0
d_shared_interests_o              0
d_attractive_important            0
d_sincere_important               0
d_intellicence_important          0
d_funny_important                 0
d_ambtition_important             0
d_shared_interests_important      0
d_attractive_partner              0
d_sincere_partner                 0
d_intelligence_partner            0
d_funny_partner                   0
d_ambition_partner                0
d_shared_interests_partner        0
interests_correlate             158
d_expected_happy_with_sd_people   0
d_like                            0
d_guess_prob_liked                0
met                             375
dtype: int64
```

class_ df의 남은 결측치

field(전공분야) : 63

interests_correlate (참가자와 파트너의 취미 상관 관계) : 158

met (이전에 만났었는지 여부) : 375

```
# met 열 결측치 최빈값으로 대체

nnnnew_df['met'] = nnnnew_df['met'].fillna(0)
```

데이터 전처리_1) class_df

interests_correlate 결측치 채우기

d_shared_interests_partner를 기준으로
interests_correlate의 평균을 찾아서 결측치를 보충

*d_shared_interests_partner : 내가 평가한 파트너의 취미

(해당 값을 높게 평가했다면 상관관계(interest_correlate)도 높을 것이므로
d_shared_interests_partner의 평균을 이용)

```
nnnew_df.groupby('d_shared_interests_partner').mean() # d_shared_interests_partner를 기준으로 평균을 찾기
```

_funny_partner	d_ambition_partner	d_shared_interests_partner	interests_correlate	d_expected_happy_with_sd_people	d_like	d_guess_prob_liked	met
0.799934	0.864869	0.504356	0.188146	1.032879	0.720697	0.915667	0.048029
0.775967	0.832485	0.516293	0.213736	1.099796	0.726578	0.892057	0.058698
0.728097	0.731118	0.380665	0.232905	0.758308	0.694864	0.987915	0.031250

0.19
0.21
0.23

```
nnnew_df.loc[(nnnew_df.d_shared_interests_partner==0)&(nnnew_df.interests_correlate.isnull()), 'interests_correlate'] =0.19
nnnew_df.loc[(nnnew_df.d_shared_interests_partner==1)&(nnnew_df.interests_correlate.isnull()), 'interests_correlate'] =0.21
nnnew_df.loc[(nnnew_df.d_shared_interests_partner==2)&(nnnew_df.interests_correlate.isnull()), 'interests_correlate'] =0.23
nnnew_df['met'] = nnnew_df['met'].fillna(0)

#d_shared_interests_partner의 값이 같은 행의 평균을 구해 interests_correlate에 결측치를 보충 (소수점 셋째자리에서 반올림)
```

데이터 전처리_1) class_df

field 결측치 채우기

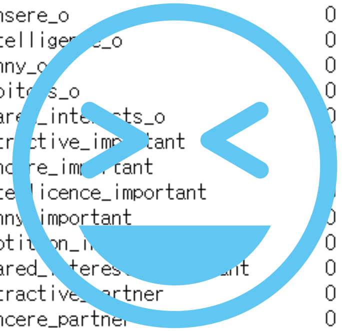
```
nnnew_df['field_new']= nnnew_df['field'].str.lower()
nnnew_df['field_new'] = nnnew_df['field_new'].str.replace("₩'", "")


field_to_5fields = {'nutritiron': 3, 'gs postbacc premed':3, 'art history':0, 'molecular biology':3, 'genetics & development':3,
                    'electrical engg.':2, 'international politics':0, 'mba / master of international affairs [sipa]':0,
                    'medicine and biochemistry':3, 'social studies education':0, 'ma teaching social studies':0, 'education policy':0,
                    'education- literacy specialist':0, 'anthropology/education':0, 'bilingual education':0, 'speech pathology':3,
                    'education':0, 'math education':0, 'tesol':0, 'cognitive studies in education':0, 'finance/economics':1,
                    'museum anthropology':0, 'environmental engineering':2, 'business administration':1,
                    'curriculum and teaching/giftedness':0, 'instructional tech & media':0, 'school psychology':0,
                    'instructional media and technology':0, 'sipa / mia':0, 'english education':0, 'ma in quantitative methods':0,
                    'early childhood education':0, 'anthropology':0, 'architecture':2, 'urban planning':0,
                    'ed.d. in higher education policy at tc':0,'human rights: middle east':0,
                    'human rights':0, 'sipa-international affairs':0, 'teaching of english':0, 'african-american studies/history':0,
                    'stats':0, 'social work/sipa':0, 'consulting':1, 'math of finance':1, 'mba - private equity / real estate':1,
nnnew_df['field_new'] = nnnew_df['field_new'].apply(lambda x: field_to_5fields.get(x, '5'))


del nnnew_df['field']
```

```
[04]: nnnew_df.isna().sum()

[04]: gender                              0
      d_d_age                             0
      samerace                            0
      d_importance_same_race              0
      d_importance_same_religion          0
      d_pref_o_attractive                 0
      d_pref_o_sincere                    0
      d_pref_o_intelligence               0
      d_pref_o_funny                      0
      d_pref_o_ambitious                  0
      d_pref_o_shared_interests           0
      d_attractive_o                      0
      d_sinsere_o                         0
      d_intelligence_o                    0
      d_funny_o                           0
      d_ambitous_o                        0
      d_shared_interests_o                0
      d_attractive_important              0
      d_sincere_important                 0
      d_intellicence_important            0
      d_funny_important                   0
      d_ambtition_important               0
      d_shared_interests_important        0
      d_attractive_partner                0
      d_sincere_partner                   0
      d_intelligence_partner              0
      d_funny_partner                     0
      d_ambition_partner                  0
      d_shared_interests_partner          0
      interests_correlate                 0
      d_expected_happy_with_sd_people     0
      d_like                              0
      d_guess_prob_liked                  0
      met                                 0
      field_new                           0
      dtype: int64
```



데이터 전처리 _2) num_df

num_df : 수치형 feature 형식 제외하고 삭제

```
del nnnnew_df ['wave']
del nnnnew_df ['has_null']
del nnnnew_df ['age']
del nnnnew_df ['age_o']
del nnnnew_df ['d_importance_same_race']
del nnnnew_df ['d_importance_same_religion']
del nnnnew_df ['d_interests_correlate']
del nnnnew_df ['d_expected_happy_with_sd_people']
del nnnnew_df ['d_like']
del nnnnew_df ['d_guess_prob_liked']
del nnnnew_df ['decision']
del nnnnew_df ['decision_o']
```

```
for d in nnnnew_df.loc[:,'d_d_age':'race_o']:
    del nnnnew_df [d]

for d in nnnnew_df.loc[:,'d_pref_o_attractive':'d_pref_o_shared_interests']:
    del nnnnew_df [d]

for d in nnnnew_df.loc[:,'d_attractive_o':'d_shared_interests_o']:
    del nnnnew_df [d]

for d in nnnnew_df.loc[:,'d_attractive_important':'d_shared_interests_important']:
    del nnnnew_df [d]

for d in nnnnew_df.loc[:,'attractive':'d_ambition']:
    del nnnnew_df [d]

for d in nnnnew_df.loc[:,'d_attractive_partner':'d_shared_interests_partner']:
    del nnnnew_df [d]

for d in nnnnew_df.loc[:,'sports':'d_yoga']:
    del nnnnew_df [d]
```

num_df

수치형을 유지한 전처리 데이터

```
nnnnew_df.head() # 수치형을 유지한 전처리 데이터
```

	gender	d_age	samerace	importance_same_race	importance_same_religion	field	pref_o_attractive	pref_o_sincere	pref_o_intelligence	pref_o_funny	pr
0	1	6	0	2.0	4.0	Law	35.0	20.0	20.0	20.0	
1	1	1	0	2.0	4.0	Law	60.0	0.0	0.0	40.0	
2	1	1	1	2.0	4.0	Law	19.0	18.0	19.0	18.0	
3	1	2	0	2.0	4.0	Law	30.0	5.0	15.0	40.0	
4	1	3	0	2.0	4.0	Law	30.0	10.0	20.0	10.0	

3

003 데이터 전처리 과정

데이터 전처리 _2) num_df

```
]: nnnnew_df.isna().sum()
```

```
]: gender                              0
d_age                               0
samerace                            0
importance_same_race               79
importance_same_religion           79
field                              63
pref_o_attractive                  89
pref_o_sincere                     89
pref_o_intelligence                89
pref_o_funny                       98
pref_o_ambitious                  107
pref_o_shared_interests           129
attractive_o                      212
sinsere_o                         287
intelligence_o                    306
funny_o                           360
ambitous_o                        722
shared_interests_o               1076
attractive_important               79
sincere_important                  79
intellicence_important             79
funny_important                    89
ambtition_important                99
shared_interests_important        121
attractive_partner                202
sincere_partner                   277
intelligence_partner              296
funny_partner                     350
ambition_partner                  712
shared_interests_partner         1067
interests_correlate               158
expected_happy_with_sd_people     101
like                              240
guess_prob_liked                  309
met                                 0
dtype: int64
```

```
7]: nnnnew_df.groupby('gender').mean()
```

gender	d_age	samerace	importance_same_race	importance_same_religion	pref_o_attractive	pref_o_sincere	pref_o_intelligence	pref
0	4.202909	0.395327	3.464542	3.096310	18.055224	18.305008	21.002502	
1	4.168260	0.396272	4.108848	4.213576	26.893883	16.497231	19.545869	

‘gender’를 기준으로 평균을 구해서 결측치 보충

```
8]: # gen...
nnnnew_df.loc[(nnnnew_df.gender==0)&(nnnnew_df.importance_same_race.isnull()), 'importance_same_race'] = 3
nnnnew_df.loc[(nnnnew_df.gender==1)&(nnnnew_df.importance_same_race.isnull()), 'importance_same_race'] = 4
nnnnew_df.loc[(nnnnew_df.gender==0)&(nnnnew_df.importance_same_religion.isnull()), 'importance_same_religion'] = 3
nnnnew_df.loc[(nnnnew_df.gender==1)&(nnnnew_df.importance_same_religion.isnull()), 'importance_same_religion'] = 4
nnnnew_df.loc[(nnnnew_df.gender==0)&(nnnnew_df.pref_o_attractive.isnull()), 'pref_o_attractive'] = 18
nnnnew_df.loc[(nnnnew_df.gender==1)&(nnnnew_df.pref_o_attractive.isnull()), 'pref_o_attractive'] = 27
nnnnew_df.loc[(nnnnew_df.gender==0)&(nnnnew_df.pref_o_sincere.isnull()), 'pref_o_sincere'] = 18
nnnnew_df.loc[(nnnnew_df.gender==1)&(nnnnew_df.pref_o_sincere.isnull()), 'pref_o_sincere'] = 16
```

field 결측치 채우기

```
nnnnew_df['field_new']= nnnnew_df['field'].str.lower()
nnnnew_df['field_new'] = nnnnew_df['field_new'].str.replace("₩'", "")


field_to_5fields = {'nutritiron': 3, 'gs postbacc premed':3, 'art history':0, 'molecular biology':3, 'genetics & development':3,
                    'electrical engg.':2, 'international politics':0, 'mba / master of international affairs [sipa]':0,
                    'medicine and biochemistry':3, 'social studies education':0, 'ma teaching social studies':0, 'education policy':0,
                    'education- literacy specialist':0, 'anthropology/education':0, 'bilingual education':0, 'speech pathology':3,
                    'education':0, 'math education':0, 'tesol':0, 'cognitive studies in education':0, 'finance/economics':1,
                    'museum anthropology':0, 'environmental engineering':2, 'business administration':1,

nnnnew_df['field_new'] = nnnnew_df['field_new'].apply(lambda x: field_to_5fields.get(x, '5'))


del nnnnew_df['field']
```

```
]: nnnnew_df.isna().sum()
```

```
]: gender                              0
d_age                               0
samerace                            0
importance_same_race                0
importance_same_religion            0
pref_o_attractive                   0
pref_o_sincere                      0
pref_o_intelligence                 0
pref_o_funny                        0
pref_o_ambitious                    0
pref_o_shared_interests             0
attractive_o                        0
sinsere_o                           0
intelligence_o                      0
funny_o                             0
ambitous_o                          0
shared_interests_o                  0
attractive_important                0
sincere_important                   0
intellicence_important              0
funny_important                     0
ambtition_important                 0
shared_interests_important          0
attractive_partner                  0
sincere_partner                     0
intelligence_partner                0
funny_partner                       0
ambition_partner                    0
shared_interests_partner            0
interests_correlate                 0
expected_happy_with_sd_people       0
like                                0
guess_prob_liked                    0
met                                 0
field_new                           0
dtype: int64
```

적용한 분석 기법 및 모델 소개

분석 기법

1) Logistic Regression
2) K-neighbors
3) Gradient Boosting Classifier
4) SVC (in Support Vector Machine)
5) Deep Learning by NN, Keras layer

Train - Test split → Create model instance variable → Train the model

1. Train - Test split

1.Train - Test split

- 범주형 데이터 트레이닝셋 테스트셋 분리

```
x1 = np.array(class_df)
y1 = np.array(y_data)

x1_train, x1_test, y1_train, y1_test = model_selection.train_test_split(x1, y1, test_size=0.1, random_state=0)
```

- 수치형 데이터 트레이닝셋 테스트셋 분리

```
x2 = np.array(num_df)
y2 = np.array(y_data)

x2_train, x2_test, y2_train, y2_test = model_selection.train_test_split(x2, y2, test_size=0.1, random_state=0)
```

```
print(x1_train.shape)
print(x1_test.shape)
print(y1_train.shape)
print(y1_test.shape)
print(x2_train.shape)
print(x2_test.shape)
print(y2_train.shape)
print(y2_test.shape)
```

```
(7540, 35)
(838, 35)
(7540, 1)
(838, 1)
(7540, 35)
(838, 35)
(7540, 1)
(838, 1)
```

2. Create model instance variable

2. Create model instance variable

Logistic Regression

```
LogisticRegression_model = linear_model.LogisticRegression()
```

K-neighbors

```
Kneighbors_model = neighbors.KNeighborsClassifier(15)
```

Gradient Boosting Classifier

```
params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
          'learning_rate': 0.01}

GradientBoostingClassifier_model = ensemble.GradientBoostingClassifier(**params)
```

SVC

```
model = svm.SVC(C=1.0, gamma='auto')
```

3. Train the model

3. Train the model

Logistic Regression

```
LogisticRegression_model.fit(x1_train, y1_train)
LogisticRegression_model.fit(x2_train, y2_train)
```

K-neighbors

```
Kneighbors_model.fit(x1_train, y1_train)
Kneighbors_model.fit(x2_train, y2_train)
```

Gradient Boosting Classifier

```
GradientBoostingClassifier_model.fit(x1_train, y1_train)
GradientBoostingClassifier_model.fit(x2_train, y2_train)
```

SVC

```
model.fit(x1_train, y1_train)
model.fit(x2_train, y2_train)
```

004 적용한 분석 기법 및 모델 소개

1) Logistic Regression

- 범주형 정확도

```
Logistic_Regression_pred_test = LogisticRegression_model.predict_proba(x1_test)
print('범주형 정확도: ', accuracy_score(LogisticRegression_model.predict(x1_test), y1_test))
```

```
범주형 정확도:  0.8687350835322196
```

- 범주형 데이터에 대한 ROC Curve

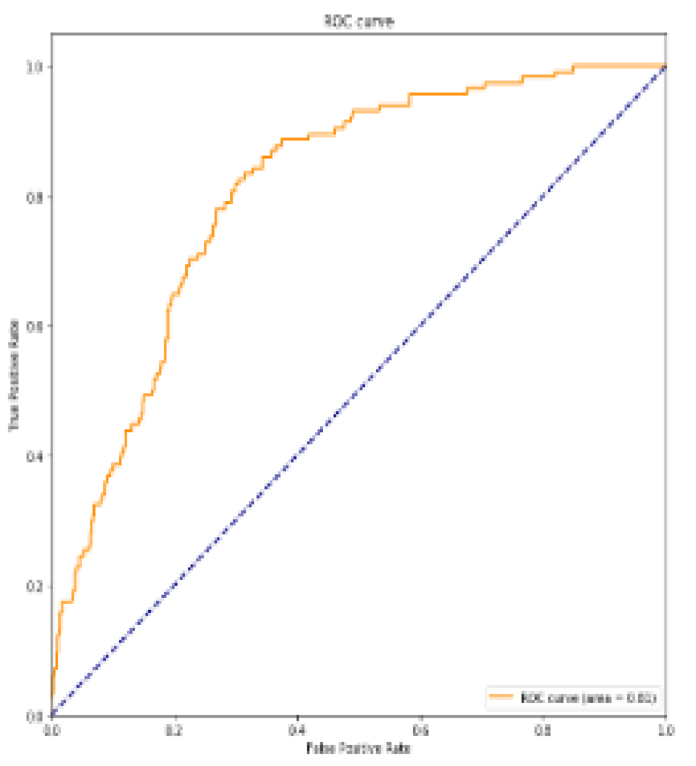
```
fpr, tpr, _ = roc_curve(y_true=y1_test, y_score=LogisticRegression_model.predict_proba(x1_test)[:,1])
roc_auc = auc(fpr, tpr) # AUC 면적의 값 (수치)
```

```
plt.figure(figsize=(10, 10))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.title("ROC curve")

plt.show()
```



1) Logistic Regression

- 수치형 정확도

```
Logistic_Regression_pred_test = LogisticRegression_model.predict_proba(x2_test)
print('수치형 정확도: ', accuracy_score(LogisticRegression_model.predict(x2_test), y2_test))
```

```
수치형 정확도:  0.8663484486873508
```

- 수치형 데이터에 대한 ROC Curve

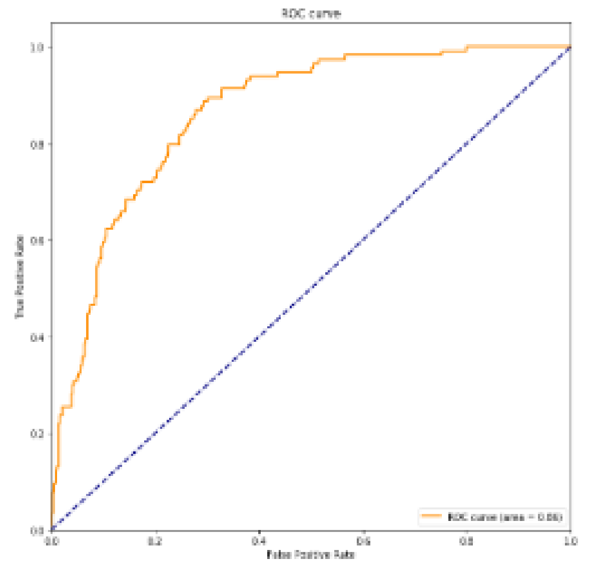
```
fpr, tpr, _ = roc_curve(y_true=y2_test, y_score=LogisticRegression_model.predict_proba(x2_test)[:,1])
roc_auc = auc(fpr, tpr) # AUC 면적의 값 (수치)
```

```
plt.figure(figsize=(10, 10))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.title("ROC curve")

plt.show()
```



2) KNN

- 범주형 정확도

```
Kneighbors_model_pred_test = Kneighbors_model.predict_proba(x1_test)
print('범주형 정확도: ', accuracy_score(Kneighbors_model.predict(x1_test), y1_test))
```

```
범주형 정확도:  0.8639618138424821
```

- 수치형 정확도

```
Kneighbors_model_pred_test = Kneighbors_model.predict_proba(x2_test)
print('수치형 정확도: ', accuracy_score(Kneighbors_model.predict(x2_test), y2_test))
```

```
수치형 정확도:  0.8663484486873508
```

3) SVM

- 수치형 정확도

```
predicted_y = model.predict(x1_test)
print('수치형 정확도 :', metrics.accuracy_score(predicted_y, y1_test))
```

```
수치형 정확도 : 0.8639618138424822
```

```
predicted_y = model.predict(x2_test)
print('수치형 정확도 :', metrics.accuracy_score(predicted_y, y2_test))
```

```
수치형 정확도 : 0.8651551312649165
```

4) Gradient Boosting Classifier

- 범주형 MSE

```
mse = mean_squared_error(y1_train, GradientBoostingClassifier_model.predict(x1_train))
print("범주형 MSE: {}".format(mse))
```

```
범주형 MSE: 0.16790450928381964
```

```
mse = mean_squared_error(y1_test, GradientBoostingClassifier_model.predict(x1_test))
print("범주형 MSE: {}".format(mse))
```

```
범주형 MSE: 0.1360381861575179
```

- 범주형 정확도

```
print('범주형 정확도: ', accuracy_score(GradientBoostingClassifier_model.predict(x1_test), y1_test))
```

```
범주형 정확도:  0.863961813842482
```

4) Gradient Boosting Classifier

- 수치형 MSE

```
mse = mean_squared_error(y2_train, GradientBoostingClassifier_model.predict(x2_train))
print("수치형 MSE: {}".format(mse))
```

```
수치형 MSE: 0.11724137931034483
```

```
mse = mean_squared_error(y2_test, GradientBoostingClassifier_model.predict(x2_test))
print("수치형 MSE: {}".format(mse))
```

```
수치형 MSE: 0.11575178997613365
```

- 수치형 정확도

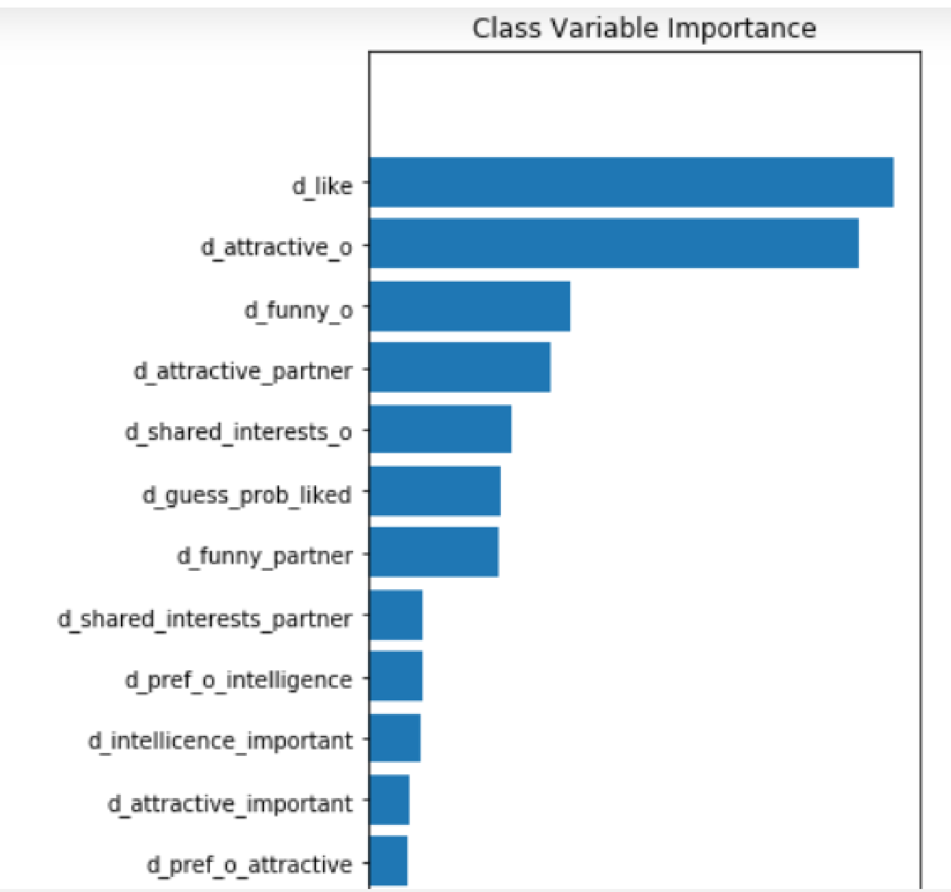
```
print('수치형 정확도: ', accuracy_score(GradientBoostingClassifier_model.predict(x2_test), y2_test))
```

```
수치형 정확도:  0.8842482100238663
```

4) Gradient Boosting Classifier

범주형 데이터 중 match와 상관관계가 높은 feature

```
feature_importance = GradientBoostingClassifier_model.feature_importances_
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize = (10, 20))
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, class_df.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Class Variable Importance')
plt.show()
```

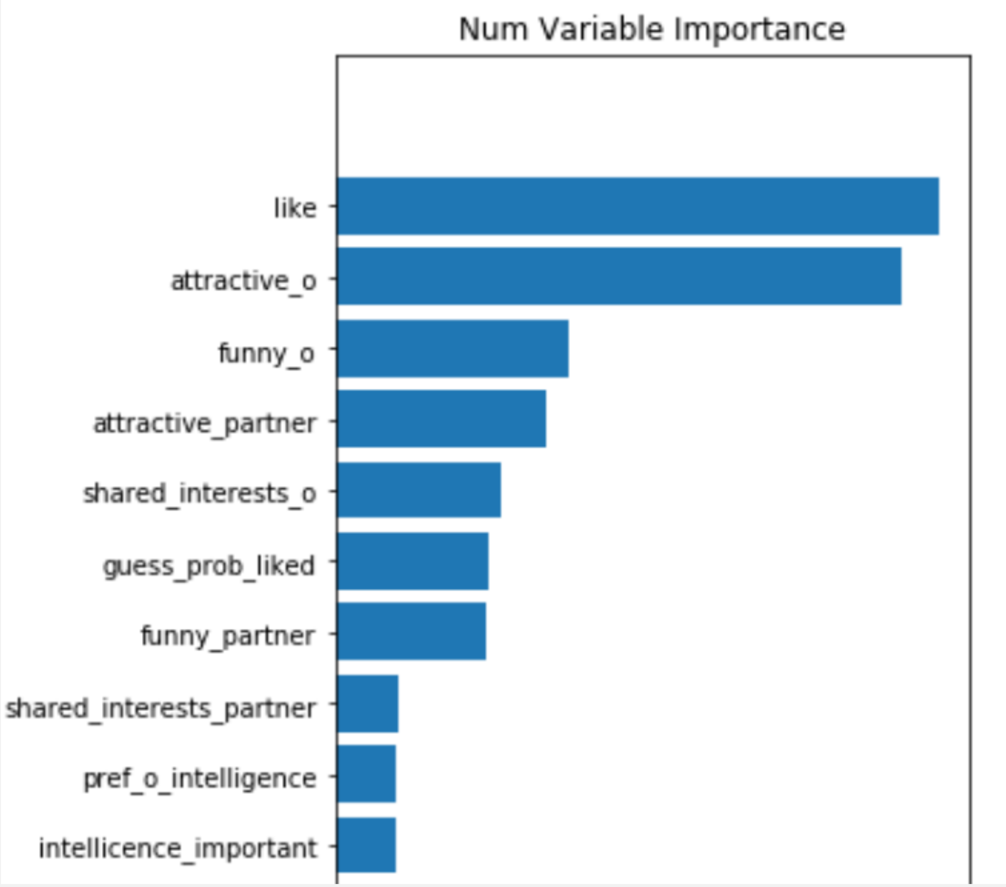


4

4) Gradient Boosting Classifier

수치형 데이터 중 match와 상관관계가 높은 feature

```
feature_importance = GradientBoostingClassifier_model.feature_importances_
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize = (10, 20))
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, num_df.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Num Variable Importance')
plt.show()
```



5) Deep Learning

```
import numpy as np
import tensorflow as tf
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # https://stackoverflow.com/questions/35911252/disable-tensorflow-debugging-information
from sklearn import model_selection
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras import datasets, utils
from tensorflow.keras import models, layers, activations, initializers, losses, optimizers, metrics
```

- 범주형 데이터 One Hot Encoding

```
enc = OneHotEncoder(categories='auto')

enc.fit(y1_train)
y1_train = enc.transform(y1_train).toarray()

enc.fit(y1_test)
y1_test = enc.transform(y1_test).toarray()

print(y1_train.shape)
print(y1_test.shape)
```

```
(7540, 2)
(838, 2)
```

```
enc = OneHotEncoder(categories='auto')

enc.fit(y2_train)
y2_train = enc.transform(y2_train).toarray()

enc.fit(y2_test)
y2_test = enc.transform(y2_test).toarray()
                                        #OneHotEncoder를 통해 타겟데이터의 feature를 2개로
print(y2_train.shape)
print(y2_test.shape)
```

```
(7540, 2)
(838, 2)
```

- 수치형 데이터 One Hot Encoding

5-1) Deep Learning by NN layer

Layer 4개 (노드갯수 : 각 35, 512, 512, 2개)
AdamOptimizer, Relu를 사용

```
X = tf.placeholder(tf.float32, [None, 35]) # [# of batch data, # of features(columns)]
Y = tf.placeholder(tf.float32, [None, 2]) # 0~9
```

```
W1 = tf.Variable(tf.random_normal([35, 512], stddev=0.01))
L1 = tf.nn.relu(tf.matmul(X, W1))
```

```
W2 = tf.Variable(tf.random_normal([512, 512], stddev=0.01))
L2 = tf.nn.relu(tf.matmul(L1, W2))
```

```
W3 = tf.Variable(tf.random_normal([512, 2], stddev=0.01))
model = tf.matmul(L2, W3)
```

```
cost = tf.losses.softmax_cross_entropy(Y, model) # for Classification
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

```
def shuffle_batch(X, y, batch_size):
    rnd_idx = np.random.permutation(len(X))
    n_batches = len(X) // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch, y_batch = X[batch_idx], y[batch_idx]
        yield X_batch, y_batch
```

5-1) Deep Learning by NN layer

범주형 데이터를 배치 100으로 설정 후 학습.

```
for epoch in range(15):
    total_cost = 0

    for batch_xs, batch_ys in shuffle_batch(x1_train, y1_train, 100):
#     for i in range(total_batch):
#         batch_xs, batch_ys = mnist.train.next_batch(100)

        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs, Y: batch_ys})
        total_cost += cost_val

    test_cost = sess.run([cost], feed_dict={X: x1_test, Y: y1_test}) # current test error

    print('Epoch: {}'.format(epoch+1),
          '|| Avg. Training cost = {:.3f}'.format(total_cost / 75),
          '|| Current Test cost = {:.3f}'.format(test_cost[0]))

print('Learning process is completed!')
```

```
Epoch: 1 || Avg. Training cost = 0.431 || Current Test cost = 0.327
Epoch: 2 || Avg. Training cost = 0.365 || Current Test cost = 0.320
Epoch: 3 || Avg. Training cost = 0.356 || Current Test cost = 0.327
Epoch: 4 || Avg. Training cost = 0.348 || Current Test cost = 0.318
Epoch: 5 || Avg. Training cost = 0.341 || Current Test cost = 0.307
Epoch: 6 || Avg. Training cost = 0.335 || Current Test cost = 0.313
Epoch: 7 || Avg. Training cost = 0.332 || Current Test cost = 0.331
Epoch: 8 || Avg. Training cost = 0.324 || Current Test cost = 0.305
Epoch: 9 || Avg. Training cost = 0.324 || Current Test cost = 0.303
Epoch: 10 || Avg. Training cost = 0.316 || Current Test cost = 0.298
Epoch: 11 || Avg. Training cost = 0.310 || Current Test cost = 0.294
Epoch: 12 || Avg. Training cost = 0.304 || Current Test cost = 0.296
Epoch: 13 || Avg. Training cost = 0.300 || Current Test cost = 0.312
Epoch: 14 || Avg. Training cost = 0.295 || Current Test cost = 0.313
Epoch: 15 || Avg. Training cost = 0.290 || Current Test cost = 0.322
Learning process is completed!
```

5-1) Deep Learning by NN layer

수치형 데이터를 배치 100으로 설정 후 학습.

```
for epoch in range(15):
    total_cost = 0

    for batch_xs, batch_ys in shuffle_batch(x2_train, y2_train, 100):
#     for i in range(total_batch):
#         batch_xs, batch_ys = mnist.train.next_batch(100)

        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs, Y: batch_ys})
        total_cost += cost_val

    test_cost = sess.run([cost], feed_dict={X: x2_test, Y: y2_test}) # current test error

    print('Epoch: {}'.format(epoch+1),
          '|| Avg. Training cost = {:.3f}'.format(total_cost / 75),
          '|| Current Test cost = {:.3f}'.format(test_cost[0]))

print('Learning process is completed!')
```

```
Epoch: 1 || Avg. Training cost = 0.752 || Current Test cost = 0.308
Epoch: 2 || Avg. Training cost = 0.350 || Current Test cost = 0.292
Epoch: 3 || Avg. Training cost = 0.358 || Current Test cost = 0.290
Epoch: 4 || Avg. Training cost = 0.344 || Current Test cost = 0.287
Epoch: 5 || Avg. Training cost = 0.335 || Current Test cost = 0.289
Epoch: 6 || Avg. Training cost = 0.342 || Current Test cost = 0.303
Epoch: 7 || Avg. Training cost = 0.331 || Current Test cost = 0.287
Epoch: 8 || Avg. Training cost = 0.331 || Current Test cost = 0.322
Epoch: 9 || Avg. Training cost = 0.327 || Current Test cost = 0.293
Epoch: 10 || Avg. Training cost = 0.328 || Current Test cost = 0.304
Epoch: 11 || Avg. Training cost = 0.326 || Current Test cost = 0.291
Epoch: 12 || Avg. Training cost = 0.329 || Current Test cost = 0.290
Epoch: 13 || Avg. Training cost = 0.328 || Current Test cost = 0.287
Epoch: 14 || Avg. Training cost = 0.320 || Current Test cost = 0.294
Epoch: 15 || Avg. Training cost = 0.322 || Current Test cost = 0.301
Learning process is completed!
```

5-1) Deep Learning by NN layer

```
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1)) # model : 예측값, Y : 실제 정답
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

- 범주형 정확도

```
print('범주형 정확도 :', sess.run(accuracy,
                            feed_dict={X: x1_test,
                                       Y: y1_test}))
```

```
범주형 정확도 : 0.40572792
```

- 수치형 정확도

```
print('수치형 정확도 :', sess.run(accuracy,
                            feed_dict={X: x2_test,
                                       Y: y2_test}))
```

```
수치형 정확도 : 0.8711217
```

5-2) Deep Learning by Keras layer

Layer 4개 (노드갯수 : 각 35, 512, 512, 2개)
AdamOptimizer, Relu를 사용
DropOut 수치 0.2

```
model = models.Sequential()

model.add(layers.Dense(input_dim=35, units=512, activation=None, kernel_initializer=initializers.he_uniform()))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu')) # layers.ELU or layers.LeakyReLU
model.add(layers.Dropout(rate=0.2))

model.add(layers.Dense(units=512, activation=None, kernel_initializer=initializers.he_uniform()))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu')) # layers.ELU or layers.LeakyReLU
model.add(layers.Dropout(rate=0.2))

model.add(layers.Dense(units=2, activation='softmax')) # 0~9
```

```
model.compile(optimizer=optimizers.Adam(),
              loss=losses.categorical_crossentropy,
              metrics=[metrics.categorical_accuracy])
```

```
enc.fit(y_data)
y_data = enc.transform(y_data).toarray()
```

5-2) Deep Learning by Keras layer

수치형 데이터를 배치 75으로
설정 후 학습

```
class_deep = model.fit(class_df, y_data, batch_size=75, epochs=15, validation_split = 0.2)
```

```
Train on 6702 samples, validate on 1676 samples
Epoch 1/15
6702/6702 [==============================] - 1s 219us/sample - loss: 0.1604 - categorical_accuracy: 0.9527 - val_loss: 0.0630 - val_cat
egorical_accuracy: 0.9845
Epoch 2/15
6702/6702 [==============================] - 2s 227us/sample - loss: 0.1208 - categorical_accuracy: 0.9596 - val_loss: 0.0655 - val_cat
egorical_accuracy: 0.9791
Epoch 3/15
6702/6702 [==============================] - 1s 217us/sample - loss: 0.1045 - categorical_accuracy: 0.9640 - val_loss: 0.0719 - val_cat
egorical_accuracy: 0.9797
Epoch 4/15
6702/6702 [==============================] - 2s 232us/sample - loss: 0.0839 - categorical_accuracy: 0.9705 - val_loss: 0.0687 - val_cat
egorical_accuracy: 0.9761
Epoch 5/15
6702/6702 [==============================] - 1s 218us/sample - loss: 0.0763 - categorical_accuracy: 0.9739 - val_loss: 0.0855 - val_cat
egorical_accuracy: 0.9755
Epoch 6/15
6702/6702 [==============================] - 1s 217us/sample - loss: 0.0729 - categorical_accuracy: 0.9757 - val_loss: 0.0759 - val_cat
egorical_accuracy: 0.9767
Epoch 7/15
6702/6702 [==============================] - 1s 217us/sample - loss: 0.0657 - categorical_accuracy: 0.9778 - val_loss: 0.0773 - val_cat
egorical_accuracy: 0.9797
Epoch 8/15
6702/6702 [==============================] - 1s 217us/sample - loss: 0.0683 - categorical_accuracy: 0.9754 - val_loss: 0.0822 - val_cat
egorical_accuracy: 0.9743
Epoch 9/15
6702/6702 [==============================] - 1s 217us/sample - loss: 0.0667 - categorical_accuracy: 0.9782 - val_loss: 0.0862 - val_cat
egorical_accuracy: 0.9773
Epoch 10/15
6702/6702 [==============================] - 2s 236us/sample - loss: 0.0612 - categorical_accuracy: 0.9787 - val_loss: 0.0899 - val_cat
egorical_accuracy: 0.9726
Epoch 11/15
6702/6702 [==============================] - 2s 227us/sample - loss: 0.0556 - categorical_accuracy: 0.9788 - val_loss: 0.0857 - val_cat
egorical_accuracy: 0.9755
Epoch 12/15
6702/6702 [==============================] - 2s 228us/sample - loss: 0.0506 - categorical_accuracy: 0.9840 - val_loss: 0.0919 - val_cat
egorical_accuracy: 0.9714
Epoch 13/15
6702/6702 [==============================] - 2s 234us/sample - loss: 0.0502 - categorical_accuracy: 0.9837 - val_loss: 0.0932 - val_cat
egorical_accuracy: 0.9737
Epoch 14/15
6702/6702 [==============================] - 1s 218us/sample - loss: 0.0473 - categorical_accuracy: 0.9824 - val_loss: 0.1109 - val_cat
egorical_accuracy: 0.9672
Epoch 15/15
6702/6702 [==============================] - 1s 222us/sample - loss: 0.0444 - categorical_accuracy: 0.9849 - val_loss: 0.0957 - val_cat
egorical_accuracy: 0.9726
```

5-2) Deep Learning by Keras layer

수치형 데이터를 배치 75으로 설정 후 학습

```
num_deep = model.fit(num_df, y_data, batch_size=75, epochs=15, validation_split = 0.2)
```

```
Train on 6702 samples, validate on 1676 samples
Epoch 1/15
6702/6702 [==============================] - 2s 232us/sample - loss: 0.3362 - categorical_accuracy: 0.8908 - val_loss: 0.8954 - val_cat
egorical_accuracy: 0.7685
Epoch 2/15
6702/6702 [==============================] - 1s 218us/sample - loss: 0.2537 - categorical_accuracy: 0.9027 - val_loss: 0.4566 - val_cat
egorical_accuracy: 0.8055
Epoch 3/15
6702/6702 [==============================] - 1s 223us/sample - loss: 0.2105 - categorical_accuracy: 0.9164 - val_loss: 0.2586 - val_cat
egorical_accuracy: 0.8842
Epoch 4/15
6702/6702 [==============================] - 1s 218us/sample - loss: 0.1777 - categorical_accuracy: 0.9303 - val_loss: 0.1906 - val_cat
egorical_accuracy: 0.9350
Epoch 5/15
6702/6702 [==============================] - 1s 221us/sample - loss: 0.1768 - categorical_accuracy: 0.9294 - val_loss: 0.2173 - val_cat
egorical_accuracy: 0.9290
Epoch 6/15
6702/6702 [==============================] - 2s 231us/sample - loss: 0.1502 - categorical_accuracy: 0.9388 - val_loss: 0.1870 - val_cat
egorical_accuracy: 0.9183
Epoch 7/15
6702/6702 [==============================] - 2s 227us/sample - loss: 0.1521 - categorical_accuracy: 0.9370 - val_loss: 0.2081 - val_cat
egorical_accuracy: 0.9177
Epoch 8/15
6702/6702 [==============================] - 1s 221us/sample - loss: 0.1496 - categorical_accuracy: 0.9423 - val_loss: 0.1939 - val_cat
egorical_accuracy: 0.9206
Epoch 9/15
6702/6702 [==============================] - 2s 227us/sample - loss: 0.1424 - categorical_accuracy: 0.9443 - val_loss: 0.2397 - val_cat
egorical_accuracy: 0.9177
Epoch 10/15
6702/6702 [==============================] - 2s 228us/sample - loss: 0.1332 - categorical_accuracy: 0.9484 - val_loss: 0.2328 - val_cat
egorical_accuracy: 0.9153
Epoch 11/15
6702/6702 [==============================] - 1s 223us/sample - loss: 0.1334 - categorical_accuracy: 0.9470 - val_loss: 0.2420 - val_cat
egorical_accuracy: 0.9141
Epoch 12/15
6702/6702 [==============================] - 2s 226us/sample - loss: 0.1322 - categorical_accuracy: 0.9496 - val_loss: 0.2254 - val_cat
egorical_accuracy: 0.9195
Epoch 13/15
6702/6702 [==============================] - 2s 249us/sample - loss: 0.1283 - categorical_accuracy: 0.9490 - val_loss: 0.2302 - val_cat
egorical_accuracy: 0.9147
Epoch 14/15
6702/6702 [==============================] - 2s 227us/sample - loss: 0.1197 - categorical_accuracy: 0.9523 - val_loss: 0.2472 - val_cat
egorical_accuracy: 0.9147
Epoch 15/15
6702/6702 [==============================] - 2s 228us/sample - loss: 0.1190 - categorical_accuracy: 0.9588 - val_loss: 0.2299 - val_cat
egorical_accuracy: 0.9141
```

5-2) Deep Learning by Keras layer

- 범주형 정확도

```
result = model.evaluate(class_df, y_data, batch_size=75)

print('loss (cross-entropy) :', result[0])
print('class test accuracy :', result[1])
```

```
8378/8378 [==============================] - 1s 60us/sample - loss: 2.6549 - categorical_accuracy: 0.8353
loss (cross-entropy) : 2.6549262318748594
class test accuracy : 0.83528286
```

- 수치형 정확도

```
result = model.evaluate(num_df, y_data, batch_size=75)

print('loss (cross-entropy) :', result[0])
print('num test accuracy :', result[1])
```

```
8378/8378 [==============================] - 1s 60us/sample - loss: 0.0906 - categorical_accuracy: 0.9735
loss (cross-entropy) : 0.09062449717402857
num test accuracy : 0.97350204
```

5-2) Deep Learning by Keras layer

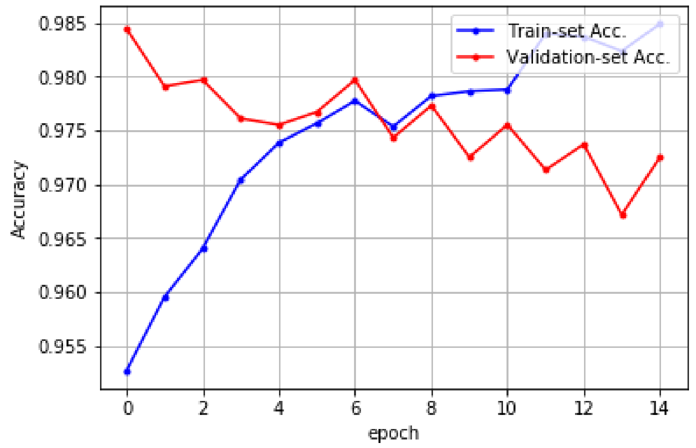
범주형 데이터 딥러닝 결과

In [521]:
```
val_acc = class_deep.history['val_categorical_accuracy']
acc = class_deep.history['categorical_accuracy']

import matplotlib.pyplot as plt

x_len = np.arange(len(acc))
plt.plot(x_len, acc, marker='.', c='blue', label="Train-set Acc.")
plt.plot(x_len, val_acc, marker='.', c='red', label="Validation-set Acc.")

plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('Accuracy')
plt.show()
```



5-2) Deep Learning by Keras layer

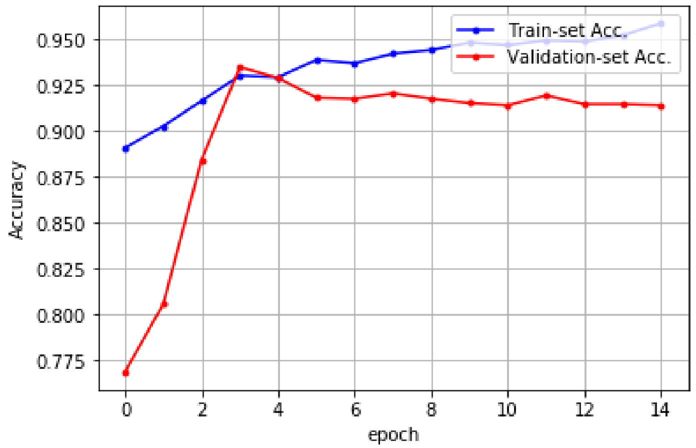
수치형 데이터 딥러닝 결과

In [522]:
```
val_acc = num_deep.history['val_categorical_accuracy']
acc = num_deep.history['categorical_accuracy']

import matplotlib.pyplot as plt

x_len = np.arange(len(acc))
plt.plot(x_len, acc, marker='.', c='blue', label="Train-set Acc.")
plt.plot(x_len, val_acc, marker='.', c='red', label="Validation-set Acc.")

plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('Accuracy')
plt.show()
```



5-2) Deep Learning by Keras layer

- Data를 집어넣으면 확률을 도출

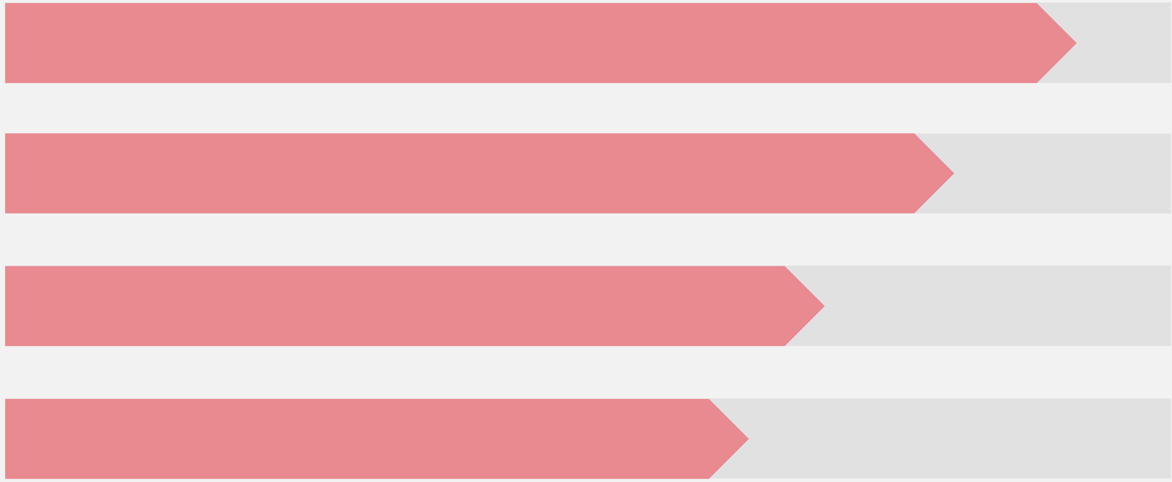
```
exp_data = np.array(num_df)
```

```
temp_data = exp_data[75, :]
temp_data = temp_data.reshape([1, 35])
predict_result = model.predict(temp_data)
predict_result *= 100
print("매칭 성공확률: " + str(predict_result[0,1]) + '%')
```

```
매칭 성공확률: 99.693695%
```

예측 결과 & Conclusion

Machine Learning 정확도



88.42% Gradient Boosting Classifier

87.59% Support Vector Machine

86.89% Logistic Regression

86.63% K – Nearest Neighbor

Deep Learning 정확도

89.41% Keras Layer

86.63% NN Layer

Match conclusion에 가장 큰 영향을 미친 요인은?

1.
파트너가 마음에 들었는가?

2.
파트너가 나를 매력적이라고 평가했는가?

3.
파트너가 재미있는가?

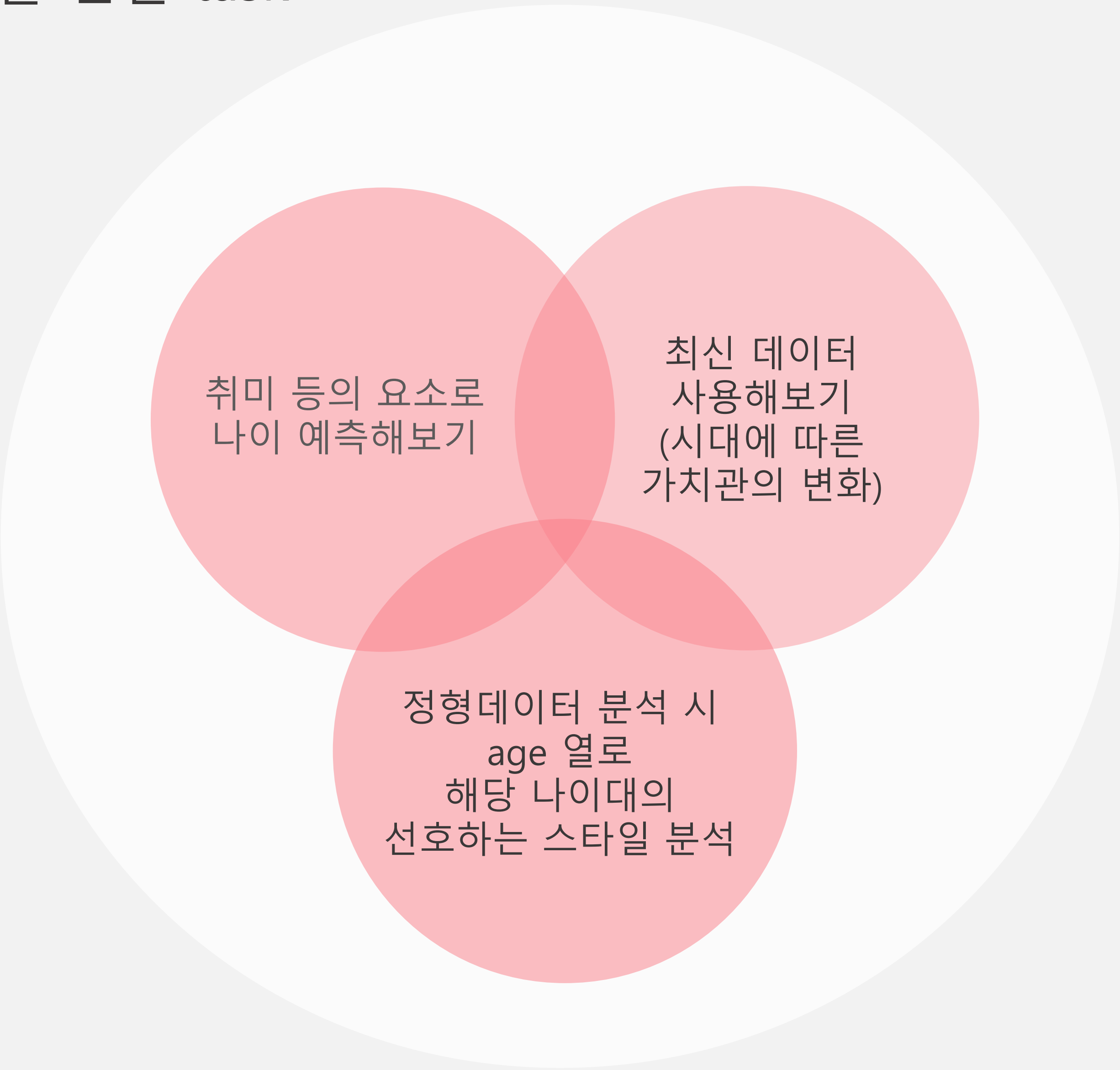
4.
파트너가 매력적인가?

5.
비슷한 취미 또는 관심사를 공유하는가?

추가로 분석해볼 만한 task

6

추가로 분석해볼 만한 task



Thank you for listening