

Base Session Wrap Up Project

MBTI 분류 및 토픽 모델링 구축 프로젝트

Boaz 19기 김보겸, 민경원, 성재혁, 이세연, 정은진

00

Index

Contents

01

Intro

02

Pre-processing

03

Classification

04

LSTM

05

Topic Modeling

06

Outro



프로젝트 주제 및 목표

- Base Session에서 배운 다양한 ML, DL 알고리즘을 프로젝트에 적용해보자!
- NLP (Natural Language Processing, 자연어처리)의 처음부터 끝까지 전과정을 주도적으로 수행해보자!

Dataset - Kaggle MBTI Types 500 Dataset

Column List

Posts (독립변수)

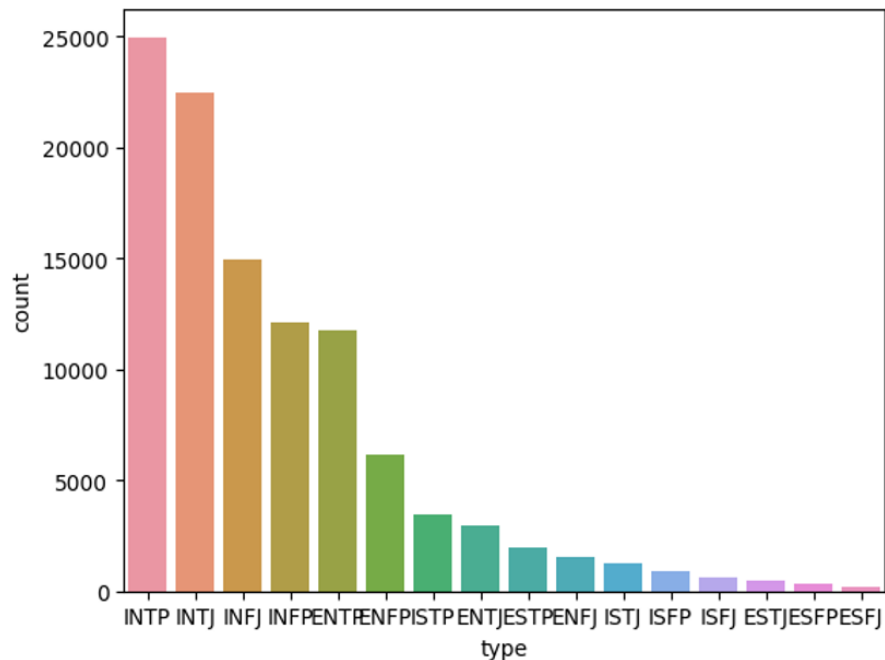
- No Punctuations, Stopwords, URLs
- Lemmatization
- Reconstruct samples to be equal-sized chunks (500 words per sample)

Type (종속변수)

- Personality types are 16 unique values

posts	type	
Preprocessed texts (500 words per sample)	Personality Type	
106067 unique values	INTP	24%
	INTJ	21%
	Other (58679)	55%
know intj tool use interaction people excuse antisocial truly enlighten mastermind know would count ...	INTJ	
rap music eh opp yeah know valid well know fact people like socialize party even personality let al...	INTJ	
preferably p hd low except wew lad video p mind good reason agree statement relationship become diff...	INTJ	
drink like wish could drink red wine give headache almost	INTJ	

EDA를 통한 데이터 분포 확인



INTP	0.235332
INTJ	0.211442
INFJ	0.141071
INFP	0.114399
ENTP	0.110543
ENFP	0.058142
ISTP	0.032281
ENTJ	0.027860
ESTP	0.018724
ENFJ	0.014463
ISTJ	0.011719
ISFP	0.008250
ISFJ	0.006128
ESTJ	0.004544
ESFP	0.003394
ESFJ	0.001706

가장 많이 등장하는 MBTI와
가장 적게 등장하는 MBTI 간의
비율은 100배 차이



데이터 불균형으로 인해
학습 시 어려움이 존재

Assumption 1. 각 경향의 독립성

- MBTI는 4가지 척도로 분류되며,
각 척도 안에는 2가지 유형 중 한 가지를 택하는 방식으로 구성

에너지 방향 (I ↔ E) / 인식 기능 (S ↔ N)
판단 기능 (T ↔ F) / 생활 양식 (J ↔ P)

- MBTI의 4가지 척도가 각각 독립적이라고 가정
 - 16개의 유형 중 1개를 예측하는 것이 아닌
각 척도 별로 어떤 유형으로 분류되는지 예측하는 문제로 변환
- 모델 종류 별로 각각 4개의 모델 구현

Assumption 2. 단어 배열의 무작위성

	posts
0	know intj tool use interaction people excuse a...
1	rap music eh opp yeah know valid well know fa...
2	preferably p hd low except wew lad video p min...
3	drink like wish could drink red wine give head...
4	space program ah bad deal meing freelance max ...
...	...
106062	stay frustrate world life want take long nap w...
106063	fizzle around time mention sure mistake thing ...
106064	schedule modify hey w intp strong wing underst...
106065	enfj since january busy schedule able spend li...
106066	feel like men good problem tell parent want te...

- 'posts' 칼럼의 단어들은 토큰화가 된 단어들
- 문맥에 맞지 않게 불규칙적으로 나열

→ word-level의 분류이므로 배열을 무작위로 섞어도 의미 손상
X

02

Pre-processing

Data Augmentation

Easy-Data-Augmentation

Operation	Sentence
None	A sad, superior human comedy played out on the back roads of life.
SR	A <i>lamentable</i> , superior human comedy played out on the <i>backward</i> road of life.
RI	A sad, superior human comedy played out on <i>funniness</i> the back roads of life.
RS	A sad, superior human comedy played out on <i>roads</i> back <i>the</i> of life.
RD	A sad, superior human out on the roads of life.

Table 1: Sentences generated using EDA. SR: synonym replacement. RI: random insertion. RS: random swap. RD: random deletion.

→ 유의어를 이용하여 문장의 일부를 바꾸는 Operation



→ 무작위의 위치에 아무 단어를 넣는 Operation

→ 문장 내 단어들의 순서를 무작위로 바꾸는 Operation



→ 문장 내 단어들을 랜덤으로 제거하는 Operation

02

Pre-processing

Data Augmentation

SR (Synonym Replacement)

```
def new_words_list(word) :
    synonymus = set()
    for words in wordnet.synsets(word) :
        for lemma in words.lemmas() :
            x = lemma.name().replace("_", "").replace("-", "").lower()
            x = "".join([char for char in x if char in 'abcdefghijklmnopqrstuvwxyz '])
            if len(x) > 2 :
                synonymus.add(x)
    if word in synonymus :
        synonymus.remove(word)
    return synonymus
```

- 3글자 이상의 단어부터 의미가 있다고 판단, 2글자 이하의 pass
- 불용어를 제외하고, 나머지 단어들에 대해 일정 비율의 유사어로 변환
 - 불용어 리스트: MBTI 유형에 해당하는 단어 (ex. ISTP, ENFJ ...)
- 서로 다른 단어가 같은 단어로 바뀌는 것에 대비해 중복 여부 확인
- wordnet의 synsets를 활용하여 유사어 목록 중의 단어를 랜덤 선정

RS (Random Swap)

```
cnt = 0
while True :
    if cnt >= cnts :
        return swap_sens
    random.shuffle(cp)
    tmps = ''.join(cp[:11])
    if tmps in dup :
        continue
    dup.add(tmps)
    swap_sens.append([' '.join(cp)])
    cnt += 1
```

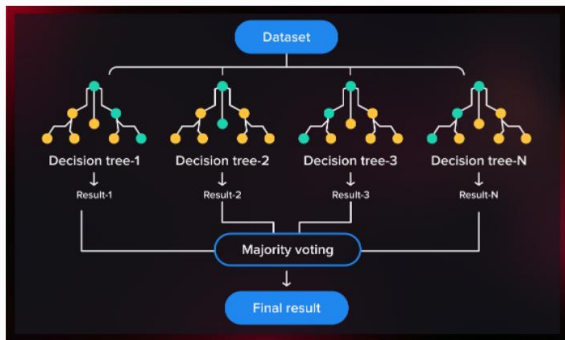
- **가정 2**에 의해 문장 내의 배열은 큰 의미가 없음을 확인
- 배열의 모든 원소를 랜덤으로 셔플 후, 문장을 생성
- 문장 중복 방지를 위해, 처음 10개의 단어 배열이 같으면 중복이라 판단
(다만 dimension 설정 값이 500이므로 문장이 중복될 확률은 매우 적음)

```
bin_class = ['i1e0', 's1n0', 't1f0', 'p1j0']
sep = ['I', 'S', 'T', 'P']
def convert_bin(types) :
    tt = []
    for k in sep :
        tt.append(int(k in types))
    return tt
data_origin[bin_class] = data_origin.apply(lambda k : pd.Series(convert_bin(k['type'])), axis = 1)
# pd.DataFrame([convert_bin('ISTJ')], columns=bin_class)
```

- 가정 1에 따라, 우리가 예측할 MBTI 척도는 총 4가지이며 각 척도 내에서 두 개의 유형 중 하나를 택하는 방식으로 진행 예정
 - 0 또는 1을 반환하도록 라벨을 Binary하게 분류
 - + Binary하게 분류한 결과를 DataFrame으로 저장, 모델링에 활용

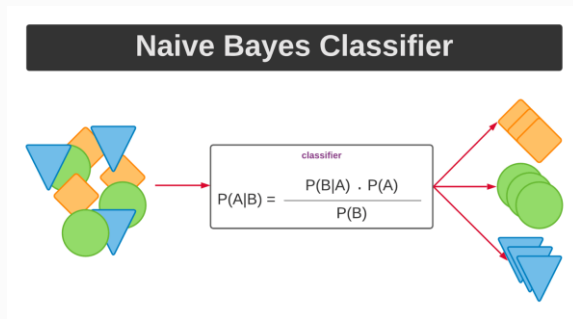
		posts	type	i1e0	s1n0	t1f0	p1j0
0	know intj tool use interaction people excuse a...	INTJ	1	0	1	0	
1	rap music ehh opp yeah know valid well know fa...	INTJ	1	0	1	0	
2	preferably p hd low except wew lad video p min...	INTJ	1	0	1	0	
3	drink like wish could drink red wine give head...	INTJ	1	0	1	0	
4	space program ah bad deal meing freelance max ...	INTJ	1	0	1	0	
...	
106062	stay frustrate world life want take long nap w...	INFP	1	0	0	1	
106063	fizzle around time mention sure mistake thing ...	INFP	1	0	0	1	
106064	schedule modify hey w intp strong wing underst...	INFP	1	0	0	1	
106065	enfj since january busy schedule able spend li...	INFP	1	0	0	1	
106066	feel like men good problem tell parent want te...	INFP	1	0	0	1	
106067 rows × 6 columns							

Random Forest



- 의사결정트리의 과적합 문제를 보완 가능
- 대표적인 앙상블 기법
- 병렬처리가 가능해 부스팅보다 빠른 처리 속도를 보이는 배깅 방법

Naive Bayes



- 베이즈 정리를 바탕으로 구현된 분류기
- 단어의 순서는 중요하지 않고, 단어의 빈도수가 중요
- 나이브 가정에 기초해 계산 속도가 빠름
- 별도의 하이퍼파라미터 튜닝이 필요하지 않음

TD-IDF

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

- TF(d, t): 특정문서 d에서 특정단어 t의 등장 횟수
- DF(t): 특정 단어 t가 등장한 문서의 수
- IDF(d,t): DF(t)에 log를 적용해 이에 반비례하는 값

TF-IDF Vectorizer

```
# Create an Tfidf Vectorizer
def vectorize(data, tfidf_vect_fit):
    X_tfidf = tfidf_vect_fit.transform(data)
    words = tfidf_vect_fit.get_feature_names()
    X_tfidf_df = pd.DataFrame(X_tfidf.toarray())
    X_tfidf_df.columns = words
    return(X_tfidf_df)

tfidf_vect = TfidfVectorizer(analyzer = 'word',
                             max_features=10000,
                             min_df = 0.2, max_df = 0.9)
tfidf_vect_fit = tfidf_vect.fit(tmp_data)

X_train = vectorize(X_train, tfidf_vect_fit)
X_test = vectorize(X_test, tfidf_vect_fit)
X_val = vectorize(X_val, tfidf_vect_fit)
```

- 특정 문서에 속한 토큰의 TF-IDF 값이 담긴 Feature Matrix 형태를 반환하는 클래스
- MemoryError 방지를 위해 max_features, min_df, max_df 설정

Naive Bayse

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score, classification_report

nb_model = GaussianNB()

nb_model.fit(X_train, y_train)

GaussianNB()

pred = nb_model.predict(X_test)
```

Before

- 파라미터 튜닝 이전, 모델 성능 측정
- Cross Validation (cv=3)

cv	Random Forest Score
1	0.88351929
2	0.8846632
3	0.88552392
Mean of RF Scores: 88.5	

Hyperparameter Tuning

```
parameters = {'n_estimators':[150,170],
              'max_depth':[20, 30],
              'max_features':[2, 5],
              'criterion':['gini','entropy']}
```

```
cv = GridSearchCV(rf, parameters)
cv.fit(X_train, y_train.values.ravel())
grid_search(cv)
```

Best Parameter	
n_estimator	170
max_depth	30
max_features	5
criterion	gini

Validation

Parameter	rf1	rf2	rf3
n_estimator	150	170	200
max_depth	30	30	20
max_features	5	5	2

Eval Metrics	rf1	rf2	rf3
Accuracy	87.4	87.4	84.3
Precision	84.3	84.6	80.4
Recall	92.7	92.4	91.7
F1 Score	88.3	88.3	85.7

Result of Prediction

Random Forest

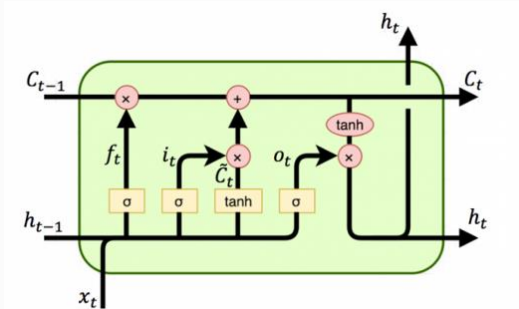
Evaluation Metrics	
Accuracy	84.2
Precision	80.4
Recall	91.6
F1 Score	85.6

Naive Bayes

Evaluation Metrics	
Accuracy	81.2
Precision	82
Recall	81
F1 Score	81

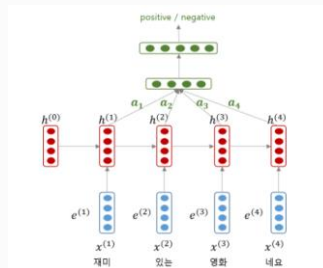
양상블 기법에 해당하는 Random Forest 모델이 Naive Bayse 모델에 비해 높은 성능을 보임

LSTM



- RNN의 기울기 소실, 장기 의존성 문제 극복을 위해 제안 됨
- cell state를 두어 과거의 정보를 효율적으로 저장
- 과거의 정보가 시간에 따라 급격하게 사라지는 것이 아닌, 일정한 비율로 사라지게 조절해 정보 유출을 규제

Attention



- seq2seq에서 제안된 메커니즘
- 디코더에서 출력 단어를 예측하는 시점마다, 인코더에서의 전체 입력 문장 다시 참고
- 전체 입력 문장을 동일한 비율로 참고 X, 해당 시점에서 예측할 단어와 연관이 있는 입력 단어에 집중

```
def tokenize_txt(data) :  
    tokenize_top_words = Tokenizer(oov_token="<OOV>", split=' ', num_words=5000)  
    fitting_text = pd.concat([data[0], data[1], data[2]])  
    # print(fitting_text)  
    tokenize_top_words.fit_on_texts(fitting_text.iloc[:, 0])  
    data_encoding = []  
    for i in data :  
        encodings = pd.DataFrame()  
        encodings['word_embedding'] = i.apply(lambda v : tokenize_top_words.texts_to_sequences([v['posts']]), axis = 1)  
        encodings['word_embedding'] = encodings.apply(lambda v: np.array(v['word_embedding']).reshape(-1).tolist(), axis = 1)  
        encodings = encodings.to_numpy().reshape(-1)  
        for j in range(len(encodings)) :  
            encodings[j] = np.array(encodings[j]).reshape(-1).tolist()  
        data_encoding.append(encodings)  
    return data_encoding
```

- Tensorflow(Keras) 내의 Tokenizer를 활용, 공백을 기준으로 단어가 분리되어 있음을 이용해 토큰나이징
- 출현 빈도를 기준으로 일정 순위 아래로는 미사용하거나 크게 의미가 없는 단어들을 판단, 상위 5000개의 단어들로 Indexing 실행
- 기본적으로 불용어 처리가 되어 있으므로 직접적인 힌트를 주는 불용어를 (미리) 제외하고 모든 단어를 tokenizing에 사용
- 추후 모델링 과정에서 Input 값으로 데이터를 입력할 것을 대비해 Tokenizing된 series를 1차원 list로 변경


```
def padding(data) :  
    padds = []  
    for i in data :  
        x = pad_sequences(i, maxlen=max_len)  
        padds.append(x)  
    return padds
```

- 불용어 처리 과정 등을 거치며 단어의 개수가 서로 달라질 수 있으므로, 학습을 진행하기 위해 모두 동일한 크기로 맞춰야 함
- 하나의 포스팅마다 단어의 최소 개수가 489, 최대 개수가 520, 평균 개수는 500임을 확인
- 따라서, 2의 거듭제곱꼴인 512로 모두 맞춘 뒤, keras의 pad_sequence 함수를 이용하여 zero padding을 실시

Tokenization

	posts	tok_tw
0	know intj tool use interaction people excuse a...	[know, intj, tool, use, interaction, people, e...
1	rap music ehk opp yeah know valid well know fa...	[rap, music, ehk, opp, yeah, know, valid, well...
2	preferably p hd low except wew lad video p min...	[preferably, p, hd, low, except, wew, lad, vid...
3	drink like wish could drink red wine give head...	[drink, like, wish, could, drink, red, wine, g...
4	space program ah bad deal meing freelance max ...	[space, program, ah, bad, deal, meing, freelan...
...
106062	stay frustrate world life want take long nap w...	[stay, frustrate, world, life, want, take, lon...
106063	fizzle around time mention sure mistake thing ...	[fizzle, around, time, mention, sure, mistake,...
106064	schedule modify hey w intp strong wing underst...	[schedule, modify, hey, w, intp, strong, wing,...
106065	enfj since january busy schedule able spend li...	[enfj, since, january, busy, schedule, able, s...
106066	feel like men good problem tell parent want te...	[feel, like, men, good, problem, tell, parent,...
106067 rows x 3 columns		

Zero-padding

	posts	tok_tw
0	know intj tool use interaction people excuse a...	[[8], [69], [888], [33], [576], [4], [1087], [...
1	rap music ehk opp yeah know valid well know fa...	[[2084], [230], [1], [1], [117], [8], [1303], ...
2	preferably p hd low except wew lad video p min...	[[1], [508], [1], [378], [554], [1], [1], [334]...
3	drink like wish could drink red wine give head...	[[361], [2], [328], [39], [361], [980], [2644]...
4	space program ah bad deal meing freelance max ...	[[509], [640], [993], [64], [234], [1], [1], [...
...
106062	stay frustrate world life want take long nap w...	[[327], [780], [81], [35], [17], [25], [70], [...
106063	fizzle around time mention sure mistake thing ...	[[1], [75], [12], [382], [62], [671], [6], [20...
106064	schedule modify hey w intp strong wing underst...	[[1252], [1], [544], [526], [73], [276], [1475]...
106065	enfj since january busy schedule able spend li...	[[512], [121], [1], [1080], [1252], [174], [21...
106066	feel like men good problem tell parent want te...	[[9], [2], [414], [14], [87], [55], [359], [17...

04

LSTM

Summary of model ①

Tokenizing

```

class BahdanauAttention(tf.keras.Model):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = Dense(units)
        self.W2 = Dense(units)
        self.V = Dense(1)

    def call(self, values, query):
        hidden_cell = tf.expand_dims(query, 1)

        score = self.V(tf.nn.tanh(self.W1(values) + self.W2(hidden_cell)))

        attention_weight = tf.nn.softmax(score, axis = 1)

        con_vec = attention_weight * values
        con_vec = tf.reduce_sum(con_vec, axis = 1)

        return con_vec, attention_weight

```

BahdanauAttention

: Attention 메커니즘이 처음으로 제시된 논문에서 사용됨

Stacked LSTM

```

lstm = LSTM(units=128, dropout=0.5, return_sequences=True)(embed_seq)
lstm, forward_h, forward_c = LSTM(units = 128,
                                   dropout = 0.5,
                                   return_sequences=True,
                                   return_state=True)(lstm)

```

- 총 2개의 Stacked Layer가 존재
- 두 층 모두 output vector의 차원이 128
- Dropout으로는 0.5를 활용
- 1층의 결과를 쌓아야 하므로, 첫층의 LSTM에서는 state를 반환하지 않음
- 2층 결과는 최종 결과에 영향을 미치므로 cell, hidden 모두 반환

04

LSTM

Summary of model ②

Decoder

```
dense1 = Dense(20, activation="relu")(con_vec)
dropout = Dropout(0.5)(dense1)
output = Dense(1, activation="sigmoid")(dropout)
```

- Attention을 활용해 얻은 128차원의 벡터를 2-Layer NN 통과
- 128 → 20 → 1 순으로 층의 차원이 진행
- 첫 번째 층에서는 가장 무난한 ReLU 활용, Dropout은 0.5
- 마지막 층에서는 분류 결과를 도출해야 하므로 sigmoid 활용

Model Combining

```
models = []
for (x, y) in zip(data_pad, dsts):
    print("=====New Data model learning start!=====")
    model = Model(inputs = sequence_input, outputs = output)
    model.compile(loss = "binary_crossentropy", optimizer='adam', metrics=['accuracy', f1score])
    X_train, y_train = x[0], y[0].iloc[:, -1]
    X_val, y_val = x[1], y[1].iloc[:, -1]
    history = model.fit(X_train, y_train, epochs = 3, batch_size = 256, validation_data=(X_val, y_val), verbose=1)
    models.append(model)
    print("=====Data model learning end!=====")
```

- (I, E) / (S, N) / (T, F) / (J, P)에 대해 각각의 모델을 구현
- 추후 결과 도출 시 편의를 위해 4개의 모델을 순서대로 list에 저장
- Batch size로는 256, Epoch는 3

04

LSTM

Result of model

Training

```
models = []
for i, (x, y) in enumerate(zip(data_pad, dsts)) :
    print("=====New Data model learning start!=====")
    model = Model(inputs = sequence_input, outputs = output)
    model.compile(loss = "binary_crossentropy", optimizer='adam', metrics=['accuracy', f1score])
    X_train, y_train = x[0], y[0].iloc[:, -1]
    X_val, y_val = x[1], y[1].iloc[:, -1]
    history = model.fit(X_train, y_train, epochs = 3, batch_size = 256, validation_data=(X_val, y_val), verbose=1)
    print("=====Data model learning end!=====")
    model.save(default_file_path + str(i))
```

✓ 258m 38.8s

=====New Data model learning start!=====

Epoch 1/3

368/368 [=====] - 1353s 4s/step - loss: 0.4112 - accuracy: 0.8189 - f1score: 0.8361 - val_loss: 0.3498 - val_accuracy: 0.8362 - val_f1score: 0.8611

Epoch 2/3

368/368 [=====] - 1442s 4s/step - loss: 0.2984 - accuracy: 0.8554 - f1score: 0.8722 - val_loss: 0.2101 - val_accuracy: 0.9152 - val_f1score: 0.9165

Epoch 3/3

368/368 [=====] - 1379s 4s/step - loss: 0.2020 - accuracy: 0.9247 - f1score: 0.9283 - val_loss: 0.1689 - val_accuracy: 0.9325 - val_f1score: 0.9352

=====Data model learning end!=====

- epoch = 3

- batch size = 256

04

LSTM

Result of model

Test

```

for i, [x, y] in enumerate(res) :
    md = tf.keras.models.load_model(default_file_path + str(i), custom_objects={"BahdanauAttention":BahdanauAttention, "f1score":f1score})
    X_test, y_test = data_pad[i][1], dsts[i][1].iloc[:, -1]
    vv = md.evaluate(X_test, y_test)
    print(f"Prediction about {x} and {y}: accuracy = {vv[1]}, f1-score: {vv[-1]}")

```

✓ 31m 35.1s

```

981/981 [=====] - 496s 504ms/step - loss: 0.1689 - accuracy: 0.9325 - f1score: 0.9337
Prediction about I and E: accuracy = 0.9325129985809326, f1-score: 0.9336780309677124
1181/1181 [=====] - 594s 502ms/step - loss: 0.0757 - accuracy: 0.9738 - f1score: 0.9719
Prediction about S and N: accuracy = 0.9737657308578491, f1-score: 0.9719204306602478
894/894 [=====] - 449s 500ms/step - loss: 0.1514 - accuracy: 0.9395 - f1score: 0.9370
Prediction about T and F: accuracy = 0.9395158290863037, f1-score: 0.9370407462120056
663/663 [=====] - 333s 500ms/step - loss: 0.2914 - accuracy: 0.8802 - f1score: 0.8928
Prediction about J and P: accuracy = 0.8802149891853333, f1-score: 0.8927884697914124

```

토픽 모델링 소개

문서 집합의 추상적 · 잠재적인 주제를
발견하기 위한 분석 방법 중 하나



어떤 단어 집합 혹은 문장이
어떤 MBTI 속성을 지녔는지
토픽을 추출하는 모델 개발

Overall Process

① MBTI가 직접적으로 언급되어 있는 단어들 제외



② 토큰화를 위해 함수 생성

1) 토큰화

2) fi, ne 등처럼 MBTI를 간접적으로 언급되어 있는 길이가 2 이하인 단어 제외

3) 품사 태깅을 통해 명사, 형용사, 동사 추출



③ MBTI별 데이터 분류

Update Stopwords

```
[ ] # MBTI 직접 언급 제외
english_stops.update(['estp', 'estj', 'esfp', 'esfj', 'entp', 'entj', 'enfp', 'enfj',
                    'istp', 'istj', 'isfp', 'isfj', 'intp', 'intj', 'infp', 'infj'])
```

- MBTI를 직접 언급하는 단어 불용어 처리

Pos tag

```
[ ] def word_pos_tag(text):
    tokens = WordFuncTokenizer().tokenize(text.lower()) # 토큰화
    words = [word for word in tokens if (word not in english_stops) and len(word) > 2] # 불용어 제거
    pos = [word for word, pos in pos_tag(words) if (pos.startswith('N')) or # 품사 태깅
           (pos.startswith('J')) or
           (pos.startswith('V'))]]
    return pos
```

- 토큰화 -> 불용어 제거 -> 품사 태깅 진행

Data Split

```
[ ] # MBTI별 데이터 분류
df_F = df[df['type'].str.contains('F')]
df_F = df_F.reset_index(drop = True)

df_T = df[df['type'].str.contains('T')]
df_T = df_T.reset_index(drop = True)

df_P = df[df['type'].str.contains('P')]
df_P = df_P.reset_index(drop = True)

df_J = df[df['type'].str.contains('J')]
df_J = df_J.reset_index(drop = True)
```

- 분류 알고리즘과 마찬가지로
- 4가지 척도 내에 존재하는 유형별로 데이터 분리
(E,I), (F,T), (P,J), (S,N)



Perplexity (O)
Coherence (X)



Sklearn
LatentDirichletAllocation


Perplexity (O)
Coherence (O)



Genism

```
[ ] dictionary_F = Dictionary(text_F)
dictionary_T = Dictionary(text_T)
dictionary_P = Dictionary(text_P)
dictionary_J = Dictionary(text_J)
print('#Number of initial unique words in documents_F: ', len(dictionary_F))
print('#Number of initial unique words in documents_T: ', len(dictionary_T))
print('#Number of initial unique words in documents_P: ', len(dictionary_P))
print('#Number of initial unique words in documents_J: ', len(dictionary_J))
```

```
#Number of initial unique words in documents_F: 116570
#Number of initial unique words in documents_T: 194034
#Number of initial unique words in documents_P: 180307
#Number of initial unique words in documents_J: 134024
```



```
[ ] # Word filtering
dictionary_F.filter_extremes(keep_n = 3000, no_below = 5, no_above = 0.5)
dictionary_T.filter_extremes(keep_n = 3000, no_below = 5, no_above = 0.5)
dictionary_P.filter_extremes(keep_n = 3000, no_below = 5, no_above = 0.5)
dictionary_J.filter_extremes(keep_n = 3000, no_below = 5, no_above = 0.5)
```

- 최대 3000개의 단어만 선정

- 최소 5번 이상 사용

- 전체 비율로는 절반(0.5)이상 사용되지 않음

→ 적절한 사용 빈도 고려

```
[ ] # 최적 토픽 수
def show_coherence(corpus, dictionary, start = 6, end = 15):
    iter_num = []
    per_value = []
    coh_value = []

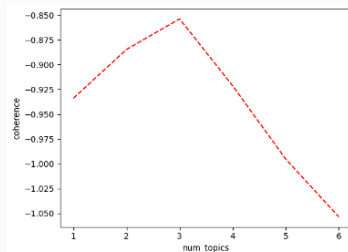
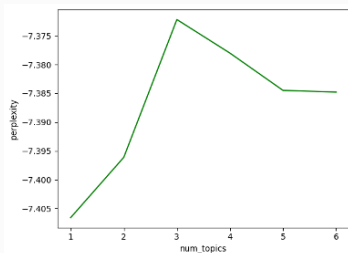
    for i in range(start, end+1):
        model = LdaModel(corpus = corpus, id2word = dictionary,
                        chunksize = 1000, num_topics = i,
                        random_state = 7)

        iter_num.append(i)
        pv = model.log_perplexity(corpus)
        per_value.append(pv)

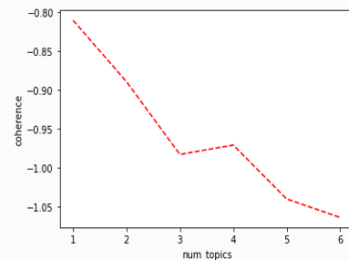
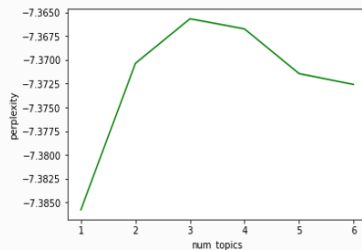
        cm = CoherenceModel(model = model, corpus = corpus, coherence = 'u_mass')
        cv = cm.get_coherence()
        coh_value.append(cv)
        print(f'num_topics: {i}, perplexity : {pv:0.3f}, coherence : {cv:0.3f}')

    plt.plot(iter_num, per_value, 'g-')
    plt.xlabel('num_topics')
    plt.ylabel('perplexity')
    plt.show()

    plt.plot(iter_num, coh_value, 'r--')
    plt.xlabel('num_topics')
    plt.ylabel('coherence')
    plt.show()
```



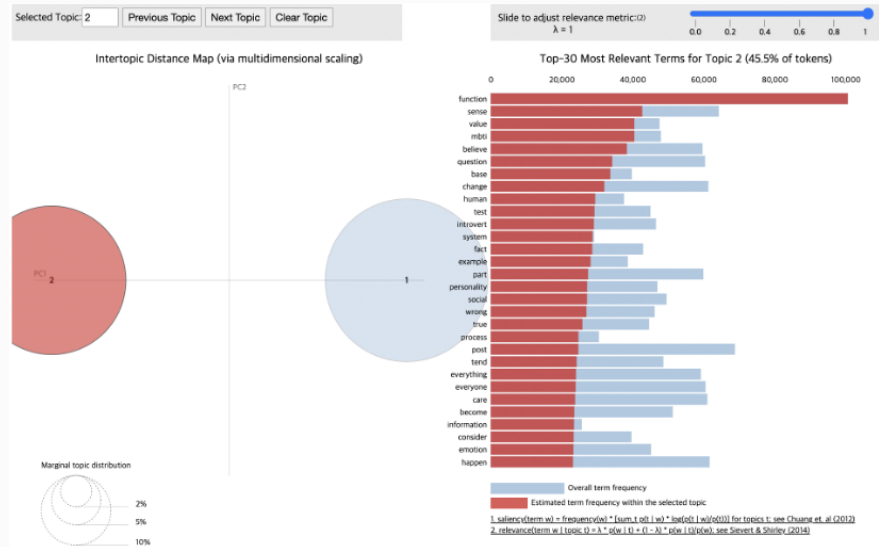
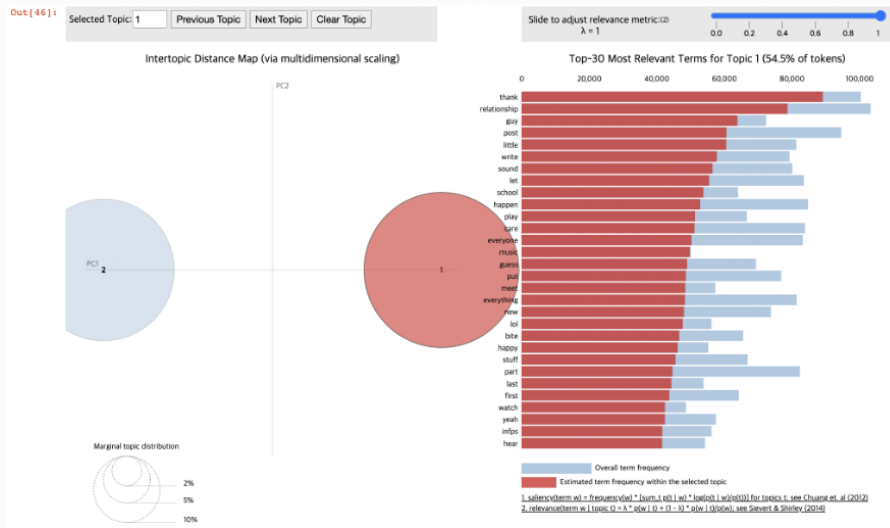
MBTI E:
perplexity 1
coherence 3



MBTI F:
perplexity 1
coherence 1

- num_topics의 최적의 perplexity와 coherence가 항상 일치할 수는 X
→ 사용자가 직접 판단
- 모든 MBTI별로 num_topics=2로 설정

pyLDAvis



Topic - Word Distribution

```
model_l.print_topics(num_words = 15)
```

Topic #

(해당 토픽에 대한 기여도)*각 토픽에 속하는 단어들

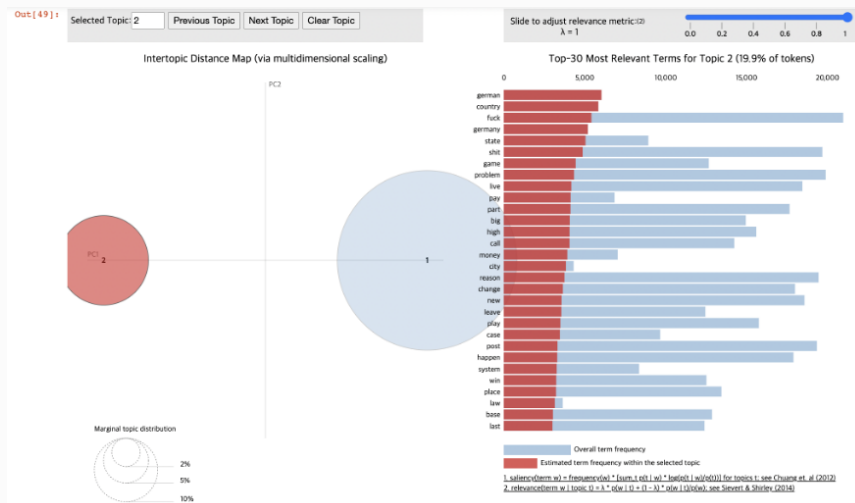
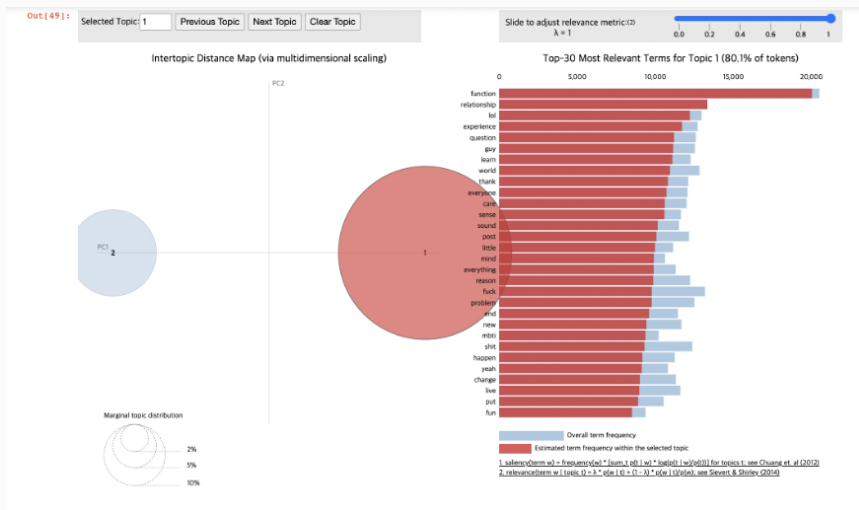
0

'0.006*"thank" + 0.005*"relationship" + 0.004*"guy" + 0.004*"post" + 0.004*"little" + 0.004*"write" + 0.004*"sound" + 0.004*"let" + 0.004*"school" + 0.004*"happen" + 0.004*"play" + 0.004*"care" + 0.003*"everyone" + 0.003*"music" + 0.003*"guess"'

1

'0.012*"function" + 0.005*"sense" + 0.005*"value" + 0.005*"mbti" + 0.004*"believe" + 0.004*"question" + 0.004*"base" + 0.004*"change" + 0.003*"human" + 0.003*"test" + 0.003*"introvert" + 0.003*"system" + 0.003*"fact" + 0.003*"example" + 0.003*"part"'

pyLDAvis



Topic - Word Distribution

```
model_E.print_topics(num_words = 15)
```

Topic #

(해당 토픽에 대한 기여도)*각 토픽에 속하는 단어들

0

'0.007*"function" + 0.004*"relationship" + 0.004*"lol" + 0.004*"experience" +
0.004*"question" + 0.004*"guy" + 0.004*"learn" + 0.004*"world" + 0.004*"thank" +
0.003*"everyone" + 0.003*"care" + 0.003*"sense" + 0.003*"sound" + 0.003*"post" +
0.003*"little"'

1

'0.005*"german" + 0.005*"country" + 0.004*"fuck" + 0.004*"germany" + 0.004*"state" +
0.004*"shit" + 0.004*"game" + 0.004*"problem" + 0.003*"live" + 0.003*"pay" +
0.003*"part" + 0.003*"big" + 0.003*"high" + 0.003*"call" + 0.003*"money"'

프로젝트 결과 요약 및 성과

- 데이터에 대한 정량적, 정성적 평가를 통해 '좋은 전처리' 방법에 대해서 고민해볼 수 있었음
- ML, DL 프로젝트를 수행하면서 발생하는 다양한 이슈를 직접 경험해볼 수 있었음
 - 데이터 불균형 상태에 대해서 인지하고, 이를 해결하기 위한 다양한 방법에 대해 탐구
 - 오버피팅 문제를 해결할 수 있는 다양한 방법 시도
- 다양한 모델의 구조에 대해서 공부해보고, 각각의 원리와 성능에 대해서 직접 테스트해볼 수 있었음
- NLP (Natural Language Processing, 자연어처리)의 처음부터 끝까지 전과정을 주도적으로 수행해보았음

- <https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/>
- <https://tyami.github.io/machine%20learning/ensemble-1-basics/>
- <https://learnopencv.com/support-vector-machines-svm/>
- <https://serokell.io/blog/random-forest-classification>
- https://yganalyst.github.io/ml/ML_chap6-2/
- <https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr>
- <https://itwiki.kr/w/LSTM>
- <https://unsplash.com/photos/zbLW0FG8XU8>, @possessedphotography
- https://github.com/strutive07/TIL/blob/master/paper_review/Neural%20machine%20translation%20by%20jointly%20learning%20to%20align%20and%20translate.md

경청해주셔서 감사합니다