영화장르 추측 챗봇

이승준

개발환경 및 사용API

- OS : Window 10
- 사용 tool : Pycharm, Jupyter-notebook
- 사용 언어 : Python
- 사용 API : TMDB API – For retrieving movie datas for using training data

  TeleGram ChatBot API – For providing user interface

소개

TMDB데이터

- 불필요한 단어 제거

Deep-Learning

- 정제된 데이터(단어)들을 이용

학습완료된 AI

- 유저로부터 받은 단어 -> 장르 유추!

코드파일 설명

https://github.com/antivec/NLP_update

-MovieDataRetriever : API를 이용하여 직접 TMDB에 접근하여 데이터 를 받아오는 코드

• NL_Processing : MovieDataRetriever를 통해 받아온 데이터의 불필요 한 요소 (symbol, number, suffix)를 제거하는 코드

• Deep_Learning : 각종 딥러닝 기법들을 이용하여 그중 가장 정확도가 높은 fit을 trained_data.txt 로 저장

• Chatbot : Telegram Chatbot API를 사용하여 trained_data.txt 를 사용 하여 유저가 입력한 단어의 장르 유사성을 측정하여 전달한다.

• movieData.csv : MovieDataRetriever로부터 받은 데이터 저장파일

• filtered_movieData.csv : NLP를 거친 데이터 저장파일

• Trained_data.txt : train 결과와 챗봇이 예측에 사용 할 데이터 파일

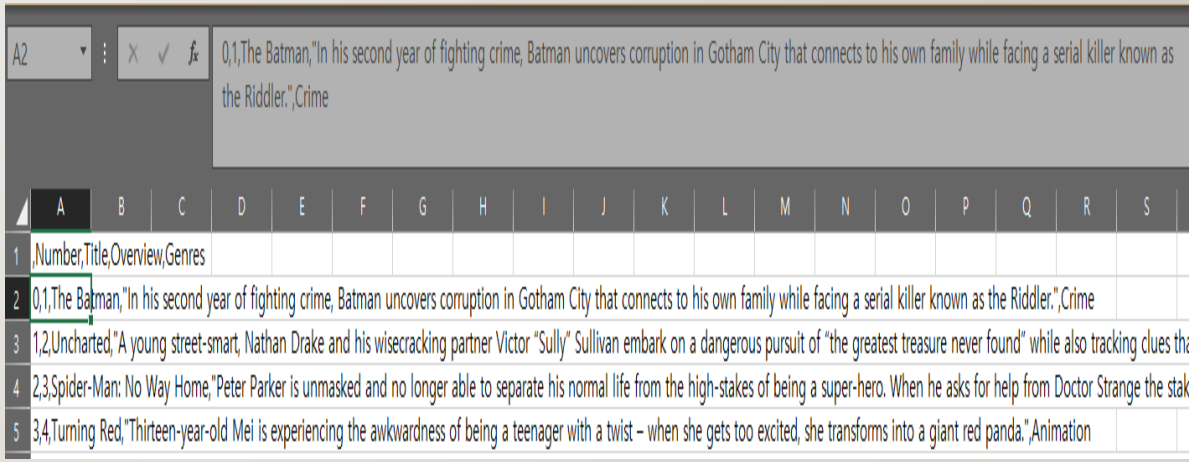
영화 정보데이터 가져오기

Https://github.com/antivec/NLP_update/MovieDataRetriver.ipynb

- TMDB api 사이트로부터 받은 키를 입력하여 DB api 사용 허가를 받음
- 약 9800여개의 영화 데이터를 csv형식으로 받아옴

[받아오는 데이터로는 타이틀, 영화 개요, 장르를 받아옴]

→영화 개요를 딥러닝을 시켜 장르 컬럼을 이용해 type화 // 개요는 영화의 줄거리와 분위기를 간략하게 알려주기에 training에 적합하다고 판단.



!! 하지만 트레이닝엔 적합하지 않은 데이터 !!

The Batman,"In his second year of fighting crime, Batman uncovers corruption in Gotham City

위 예시 문장과 같이 “THE“ ,“IN”,“HIS”,“OF” 는 영화장르의 성격과 특징을 설명하기엔 어렵다!
→ 무의미한 단어 및 문장부호 삭제 필요 // 이를 통해 기대할 수 있는 효과

1. 영화장르 성격과 특징을 특정할 수 있는 데이터 선정

2. 무의미한 단어 & 문장부호 (features) 삭제로 인한 트레이닝 속도 상승

3. Max_features를 미리 감소 시켜 모델 성능의 최적화

NL_PROCESSING

- 트레이닝에 불필요한 단어 및 쉼표,따옴표 등의 문장부호 제거
- #Stemming : chops off the ends of words • [to -> x, an -> x, a -> x]
- 예) 장르 :Action,A boy is given the ability to becoming an adult superhero in times of need with a single magic word.
- • After Stemming –> boy given ability becoming adult superhero times need single magic word
- # Lemmatizing : remove inflectional endings (ed, ing , s, es…..) only and to return the base or dictionary form of a word • # [are ,is -> be], [given -> give],
- After Lemmatizing → boy give ability become adult superhero time need single magic word

In [33]:
```
def NLP(input_data):
    lemmatize = []
    lemma = WordNetLemmatizer()
    lower = ""
    total_data_num = len(input_data.Number)
    tmp_stops = set(stopwords.words('english'))

    for i in range(total_data_num):
        stemmed = []
        items = input_data.Overview[i]
        if type(items) == float:
            items = "none"
        else:
            filtering_symbols= re.findall(r'₩b[a-z,A-Z]{2,15}₩b', items)
            max_num = len(filtering_symbols)
            for j in range(0,max_num):
                tokens = filtering_symbols[j].lower()
                if  tokens not in tmp_stops:
                    stemmed.append(tokens)

            filtered = [lemma.lemmatize(w,'v') for w in stemmed]
            sentence = ' '.join(filtered)
            input_data.Overview[i] = sentence


    return input_data
```

- Lemmatizing, StopWord 규칙을 변수로 미리 저장

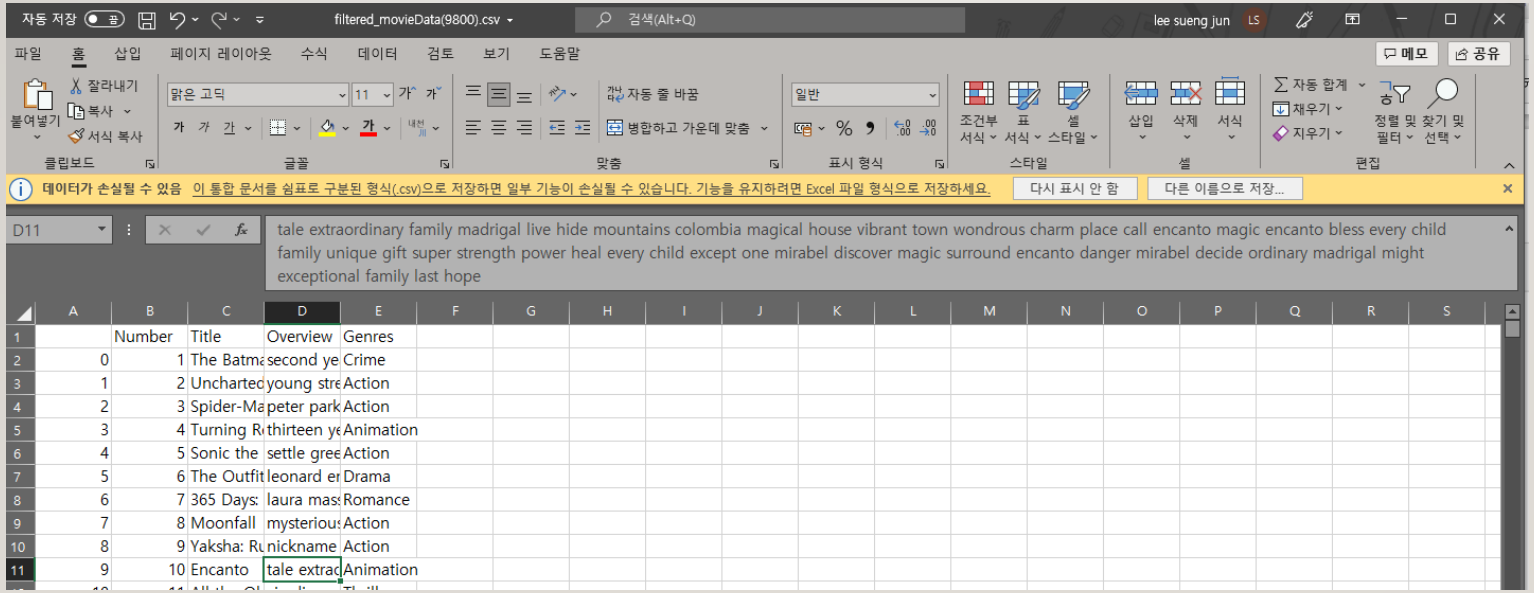
- Csv 데이터를 처음부터 끝까지 사용
- Re (python 내부함수 regular expression)

→ 혹시라도 깨진 단어를 정제하지 않도록 예외처리.

TRAINING에 사용할 준비가 된 데이터는 또 다른 CSV 파일에 저장한다

The Batman,"In his second year of fighting crime, Batman uncovers corruption in Gotham City

second year fight crime batman uncover corruption gotham city connect family face serial killer know riddler



딥러닝 기법 선정

- Vectorizer = tf – idf vectorizer
- 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타낼 수 있다.
- Classifier → Kneighbor & logistic regression(multinomial)
- Kneighbor : 단어와 특정 카테고리의 데이터 집합과의 거리 계산
- Logistic regression:카테고리 집합의 평균값에 가깝도록 "한 단어"의 값을 거꾸로 추측
- 두개의 classifier를 사용하여 둘 중의 스코어가 가장 normal에 가까운 값을 사용

장르 벡터라이즈 (KFOLD)

- 데이터 덩어리를 각 장르마다 나누고 그중 데이터 중 10개의 분할로 일정 횟수의 훈련을 통해 검증 점수를 산정한다.
- 이는 fitting단계에서 training / test에 필요한 기준점이 될 것이다
- 빨간색 칸은 kfold 설정
- 파란색 칸은 dataframe genre column 분리 작업 과 train 및 test 실행
- 노란색 칸은 result를 저장한다

```
for genre in genres[:9]:
    data['genre_y'] = [1 if y == genre else 0 for y in data['Genres']]
    k = 1
    for vect_name, vect in vectorizer.items():
        for clf_name, clf in classifier.items():
#           print(genre.upper() + ' [' + str(k) + '/' +str(n_iter) + ']: ' + vect_name + ' - ' + clf_name )
            kf = KFold(n_splits=10, random_state=None, shuffle=False)
            pipe = Pipeline([('vect', vect), ('clf', clf)])

            acc = []
            prec = []
            rec = []
            f1 = []
            for train_index, test_index in kf.split(data):
                x_train, y_train = data.Overview.iloc[train_index], data.genre_y.iloc[train_index]
                x_test, y_test = data.Overview.iloc[test_index], data.genre_y.iloc[test_index]
                pipe.fit(x_train, y_train)
                y_pred = pipe.predict(x_test)
                acc.append(np.mean(y_pred==y_test))
                prec.append(precision_score(y_test, y_pred))
                rec.append(recall_score(y_test, y_pred))
                f1.append(f1_score(y_test, y_pred))
```

변수명 // vect (벡터라이즈된 값이 저장된 dict)
CLF(classifier 가 저장된 dict 자료형)

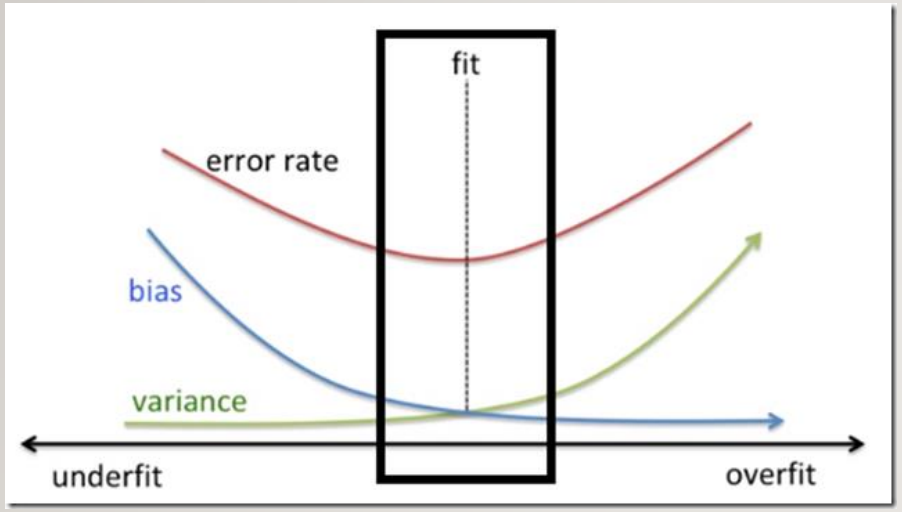
FITTING

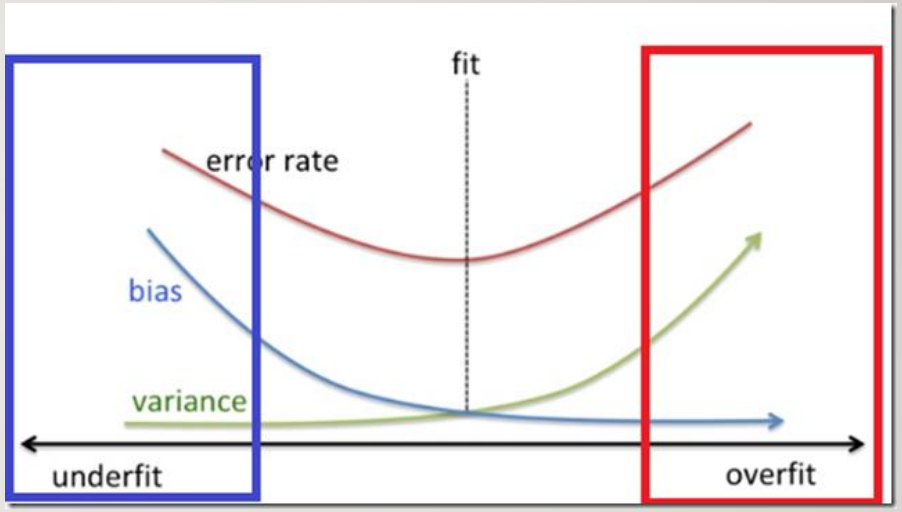
- 파란색 박스 : 앞서 vectorize가 완료된 train_vect를 사용하여 본격적으로 {classifier : KNN & Logistic} 을 사용하여 fitting을 시작한다.
- 빨간색 박스 : 추가적으로 SMOTE와 EditedNearestNeighbours 기법을 사용해 잘못된 결과가 나올수 있는 경우를 줄여준다
- 노란색 박스: FINAL_SCORE DICT자료형에 저장된 NORMAL_FIT의 점수들중 가장 낮은 점수를 추출한다

```
            clf.fit(train_vect, y_train)

            test_vect = vect.transform(x_test)
            y_pred = clf.predict(test_vect)

            acc_normal.append(np.mean(y_pred==y_test))
            prec_normal.append(precision_score(y_test, y_pred))
            rec_normal.append(recall_score(y_test, y_pred))
            f1_normal.append(f1_score(y_test, y_pred))
            auc_normal.append(roc_auc_score(y_test, y_pred))

            train_vect_over, y_train_over = SMOTE().fit_resample(train_vect, y_train)
            clf.fit(train_vect_over, y_train_over)

            test_vect = vect.transform(x_test)
            y_pred = clf.predict(test_vect)

            acc_over.append(np.mean(y_pred==y_test))
            prec_over.append(precision_score(y_test, y_pred))
            rec_over.append(recall_score(y_test, y_pred))
            f1_over.append(f1_score(y_test, y_pred))
            auc_over.append(roc_auc_score(y_test, y_pred))
            #print('Overfitting: iteration ' + str(i) + ': ')

            train_vect_under, y_train_under = EditedNearestNeighbours().fit_resample(train_vect, y_train)
            clf.fit(train_vect_under, y_train_under)

            test_vect = vect.transform(x_test)
            y_pred = clf.predict(test_vect)

            acc_under.append(np.mean(y_pred==y_test))
            prec_under.append(precision_score(y_test, y_pred))
            rec_under.append(recall_score(y_test, y_pred))
            f1_under.append(f1_score(y_test, y_pred))
            auc_under.append(roc_auc_score(y_test, y_pred))
            i+=1
        #final_score[(vect_name, clf_name, 'over')] = np.mean(auc_over)
        final_score[(vect_name, clf_name, 'normal')] = np.mean(auc_normal)
        #final_score[(vect_name, clf_name, 'under')] = np.mean(auc_under)
        #print(final_score.keys())
    test_result[genre] = min(final_score, key=final_score.get)
    print(genre , test_result[genre])
```

```
Action ('tfidf', 'logistic_regression', 'normal')
Drama ('tfidf', 'logistic_regression', 'normal')
Comedy ('tfidf', 'logistic_regression', 'normal')
```

OVERFIT?? UNDERFIT??





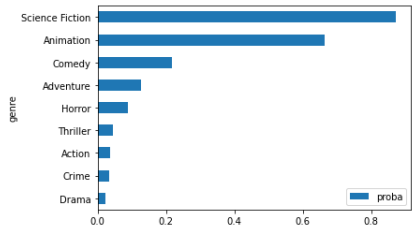
UNDERFIT은 트레이닝에 사용된 FEATURE수가 적어 실제 값이 떨어지고
OVERFIT은 트레이닝에 사용된 FEATURE수가 너무 많아 예측 값이 떨어진다
따라서 그 중앙에 ERROR 값이 제일 적은 NORMAL FIT을 도출하는 MODEL을 사용해야 한다

```
            k+=1
            #final_score[(vect_name, clf_name, 'over')] = np.mean(auc_over)
            final_score[(vect_name, clf_name, 'normal')] = np.mean(auc_normal)
            #final_score[(vect_name, clf_name, 'under')] = np.mean(auc_under)
            #print(final_score.keys())
    test_result[genre] = min(final_score, key=final_score.get)
```

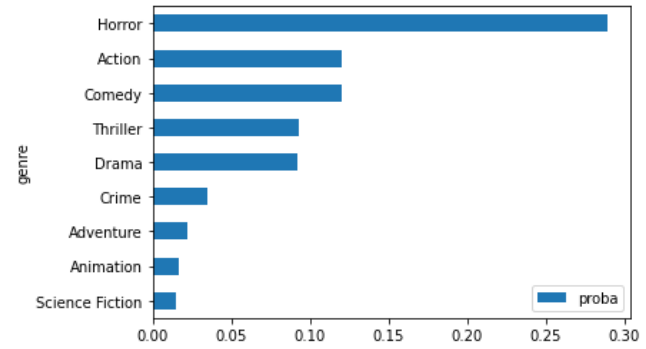
OVERFIT(왼쪽) 모델 VS NORMAL FIT(오른쪽) 모델

(OVERFIT의 모델이 이상한 추측분포비율을 보여주지만,
NORMAL FIT의 모델의 경우 전체적으로 어느정도 균등해진것을 확인 할 수 있었다)

```
In [92]: predict_genre("machine", pipe_dict)

Action Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', CalibratedClassifierCV
())])
Action
Drama Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', CalibratedClassifierCV
())])
Drama
Comedy Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', LogisticRegression())])
Comedy
Horror Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', LogisticRegression())])
Horror
Animation Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', LogisticRegression
())])
Animation
Adventure Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', CalibratedClassifierC
V())])
Adventure
Thriller Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', LogisticRegression
())])
Thriller
Crime Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', LogisticRegression())])
Crime
Science Fiction Pipeline(steps=[('vect', TfidfVectorizer()), ('clf', LogisticRegress
ion())])
Science Fiction
['Action', 'Drama', 'Comedy', 'Horror', 'Animation', 'Adventure', 'Thriller', 'Crim
e', 'Science Fiction'] [0.036872410232040825, 0.022096494833300596, 0.21669277303260
59, 0.09010956873389427, 0.663149302441221 8, 0.1277268198412911, 0.04507732736585738
6, 0.035282088561537965, 0.8729152905173471]
```




```
In [10]: predict_genre("dead", pipe_dict)

dead  :  Horror 28.99 %
```



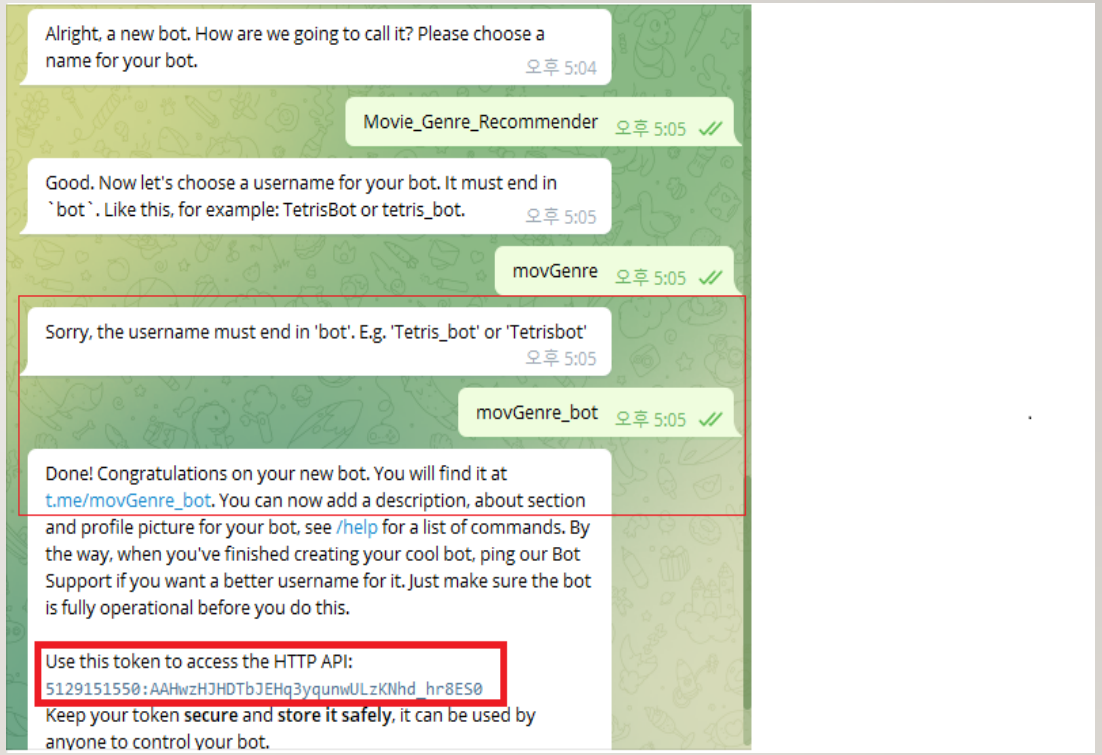
챗봇 환경 구축

https://github.com/antivec/NLP_update/Chatbot.py

-챗봇을 만드는 과정의 캡쳐이다

-텔레그램은 BotFather를 통해 메신저에서 사용자가 원하는 유용한 봇 생성이 가능하도록 해준다.

-생성에 성공하면 api 키를 제공해준다



챗봇 동작 코드 구축(I)

- 이 코드가 시작하는 동시에 상호작용할 유저의 id를 저장하는 .txt파일을 만들고, 트레이닝된 데이터가 저장된 .txt를 불러온다.

- 실행이 지속되는 동안 유저로부터 받은 메시지와 id를 업데이트하고 상호작용을 한다.

```
if __name__ == '__main__':
    try:
        with open('last_update_id.txt', 'r') as file:
            next_update_id = int(file.read())
        with open('trained_data(normal_fit).txt', 'rb') as f:
            trained = pickle.load(f)
    except (FileNotFoundError, ValueError):
        next_update_id = 0

    while True:
        last_update_id = check_messages_and_response(next_update_id, trained)
        if last_update_id:
            next_update_id = last_update_id + 1
            with open('last_update_id.txt', 'w') as file:
                file.write(str(next_update_id))
        time.sleep(2)
```

챗봇 동작 코드 구축 (II)

- BotFather로부터 부여받은 api를 키를 bot_token변수에 저장하였다.

- 챗봇으로 사용자의 채팅 데이터는
- json파일 형식 -> dict 자료형으로 받아오는데 이를 해석하기 위해 simplify_messages 함수를 생성하였다.

```
bot_token = '5129151550:AAHwzHJHDTbJEHq3ygunwULzKNhd_hr8ES0'

def request(url):
    response = urllib.request.urlopen(url)
    byte_data = response.read()
    text_data = byte_data.decode()
    return text_data

def build_url(method, query):
    return f'https://api.telegram.org/bot{bot_token}/{method}?{query}'

def request_to_chatbot_api(method, query):
    url = build_url(method, query)
    response = request(url)
    return json.loads(response)

def simplify_messages(response):
    result = response['result']
    if not result:
        return None, []
    last_update_id = max(item['update_id'] for item in result)
    messages = [item['message'] for item in result]
    simplified_messages = [{'from_id': message['from']['id'],
                            'text': message['text']}
                           for message in messages]
    return last_update_id, simplified_messages
```

챗봇 동작 코드 구축(III)

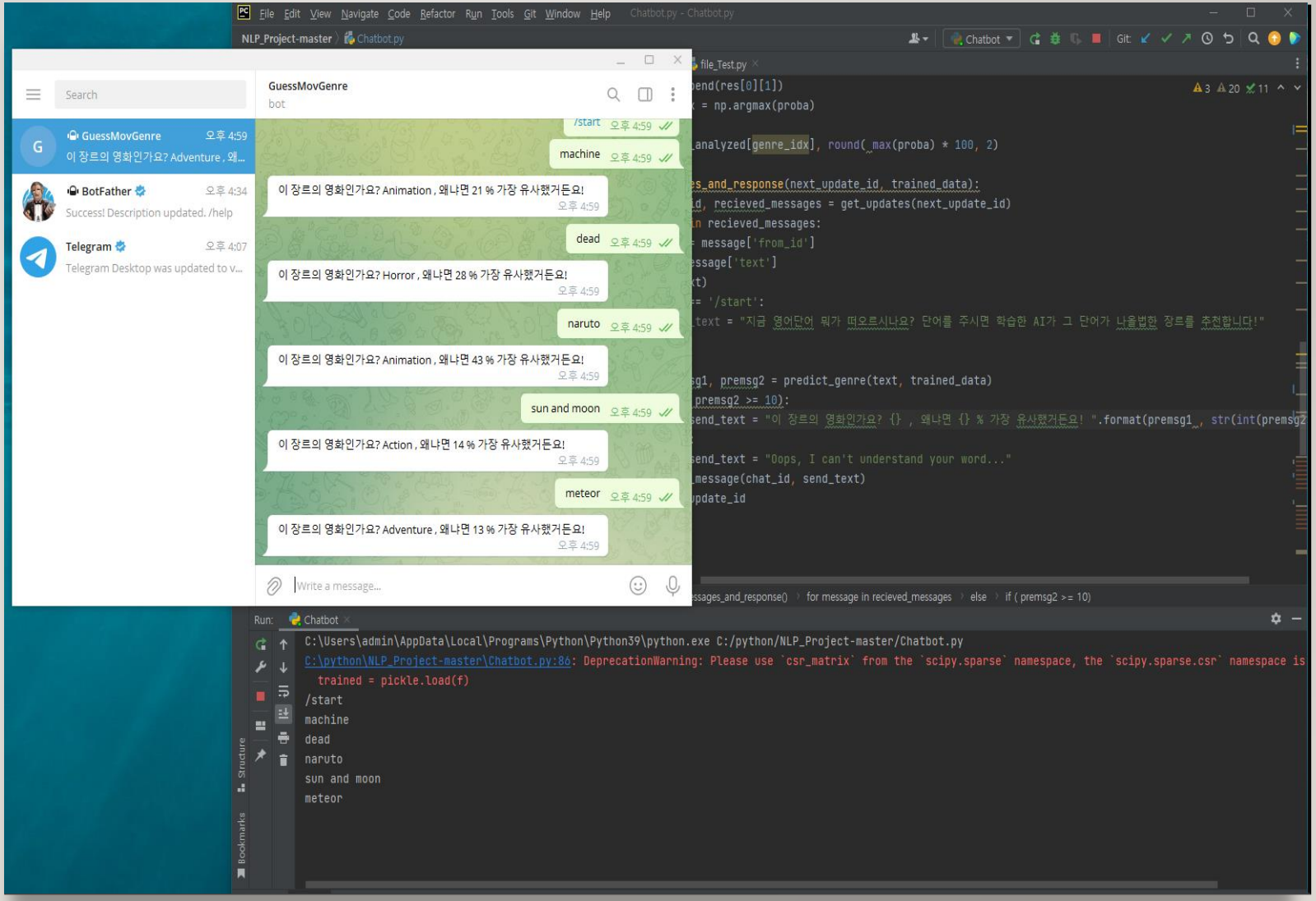
- Predict_genre 함수는 (유저로부터 받은 string, 앞서 트레이닝한 pipeline) 의 매개변수를 사용한다.
  → Return 값으로는 장르명과 결과값을 돌려준다.

- Check_messages_and_response 함수는 Predict_genre함수로부터 받은 결과로 유저의 메시지를 응답한다.

- 결과값 10%를 기준으로 그 미만의 단어들은 트레이닝하기에 표본값이 적어 딥러닝을 할 수 없었기에 유저에게 알수없다는 메시지를 전송한다.

```python
def predict_genre(s, pipe_dict):
    # s_new = clean_sentence(s)
    genre_analyzed = []
    proba = []
    for genre, pipe in pipe_dict.items():
        res = pipe.predict_proba([s])
        genre_analyzed.append(genre)
        proba.append(res[0][1])
        genre_idx = np.argmax(proba)

    return genre_analyzed[genre_idx], round(max(proba) * 100, 2)

def check_messages_and_response(next_update_id, trained_data):
    last_update_id, recieved_messages = get_updates(next_update_id)
    for message in recieved_messages:
        chat_id = message['from_id']
        text = message['text']
        print(text)
        if text == '/start':
            send_text = "지금 영어단어 뭐가 떠오르시나요? 단어를 주시면 학습한 AI가 그 단어가 나올법한 장르를 추천합니다!"

        else:
            premsg1, premsg2 = predict_genre(text, trained_data)
            if ( premsg2 >= 10):
                send_text = "이 장르의 영화인가요? {} , 왜냐면 {} % 가장 유사했거든요! ".format(premsg1, str(int(premsg2)))
            else:
                send_text = "Oops, I can't understand your word..."
            send_message(chat_id, send_text)
    return last_update_id
```

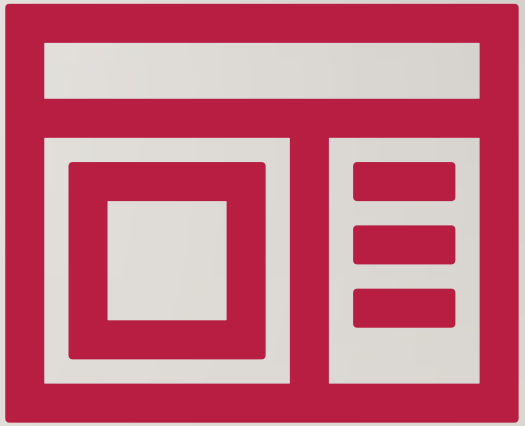
최종 봇 실행 캡처


GuessMovGenre
bot
machine
이 장르의 영화인가요? Animation , 왜냐면 21 % 가장 유사했거든요!
dead
이 장르의 영화인가요? Horror , 왜냐면 28 % 가장 유사했거든요!
naruto
이 장르의 영화인가요? Animation , 왜냐면 43 % 가장 유사했거든요!
sun and moon
이 장르의 영화인가요? Action , 왜냐면 14 % 가장 유사했거든요!
meteor
이 장르의 영화인가요? Adventure , 왜냐면 13 % 가장 유사했거든요!
Write a message...
Run: Chatbot
C:\Users\admin\AppData\Local\Programs\Python\Python39\python.exe C:/python/NLP_Project-master/Chatbot.py
/start
machine
dead
naruto
sun and moon
meteor

끝맺음

앞으로의 개선 방안

- 영화 데이터 뿐만 아닌, 챗봇으로서 사용자와 직접 소통할 수 있는 기회가 생겼기에, 유저로 부터 받은 text를 추가적인 training & test data로 정제하여 사용 → 더욱 정밀한 모델 개선
- 추가적인 기능 설계 → 유저로부터 장르를 받으면 그 장르의 분위기에 맞는 영화 제목 랜덤 생성 후 제공
- → 랜덤으로 특정 장르에 해당하는 영화 추천 및 영화예매 사이트 연결



후기

- 실제로 딥러닝 예측 모델을 만들기 전엔 많은 데이터셋이 절대적으로 필요하다고 생각했지만 실제로는 적당한 양 (overfit, underfit)과 데이터의 분류의 연관성과 정확성 또한 고려해야 하는 것을 알게 되었다.

감사합니다!