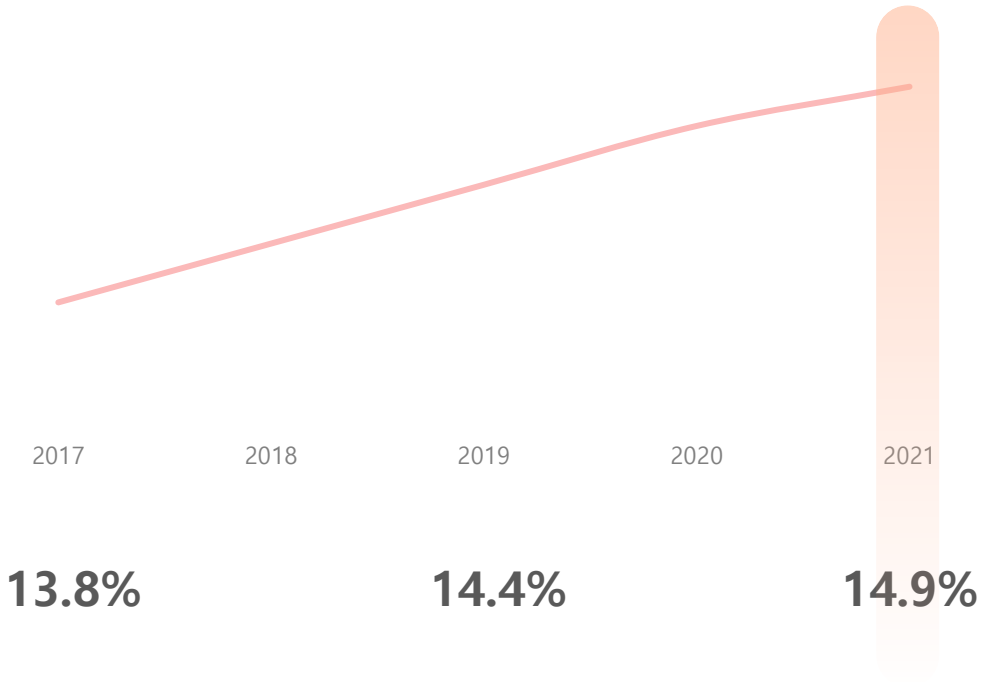
퍼스널 컬러 진단 프로그램 COLOR-ME

KDT 5기 최은아

CONTENTS

프로젝트 개요

국내 화장품 시장 연도별 시장 규모 (단위 : 조)

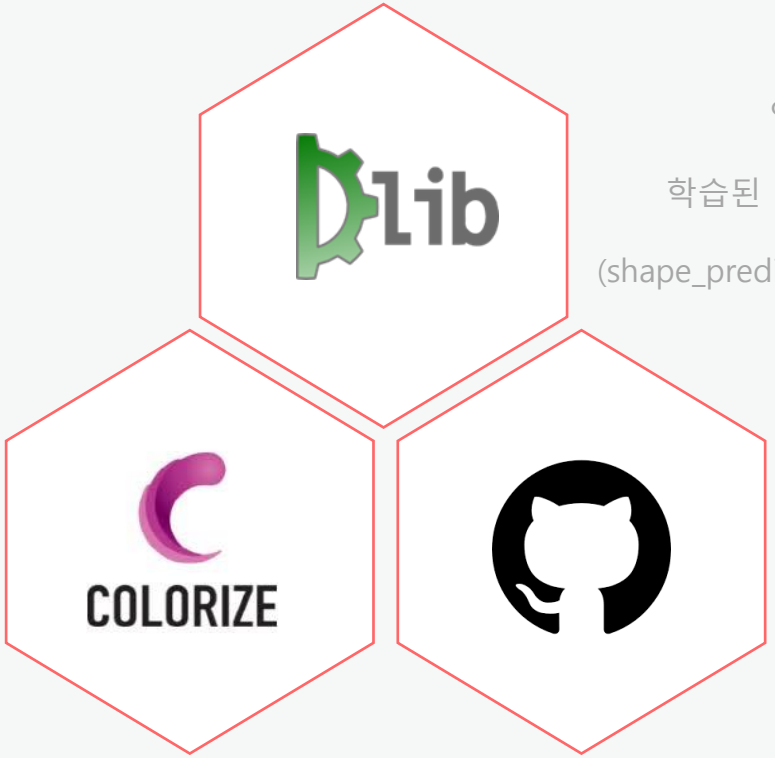


출처 : 삼성증권, Euromonitor

최근 뷰티 산업 및 sns의 활성화로 인해
아름다워지고자 하는 인간의 욕구가 과거보다
증대되면서, 자신에게 어울리는 메이크업을
찾고자 하는 경향이 강해지고 있다.
이에 따라 자신을 돋보이게 하는 퍼스널 컬러가
주목받으면서 전문가에게 자신의 퍼스널 컬러를
진단 받는 사람이 늘어나고 있다.
하지만 이는 시간적, 비용적 소모가 발생한다는 문제
점이 있다.
따라서 이러한 문제점을 해소한 퍼스널 컬러 진단 및
화장품 추천 서비스를 제공하고자 한다.

개발 언어 및 라이브러리, 프레임워크

데이터셋



dlib.net/files/

얼굴 인식을 위해

학습된 랜드마크 모델 가져오기

(shape_predictor_68_face_landmarks.dat)

COLORIZE

퍼스널 컬러 진단 사이트인 COLORIZE의

Beauty Patch 커뮤니티에서

퍼스널 컬러 별

화장품 추천 게시글 크롤링

GitHub

챗봇 구현에 사용할 ChatbotData.csv는

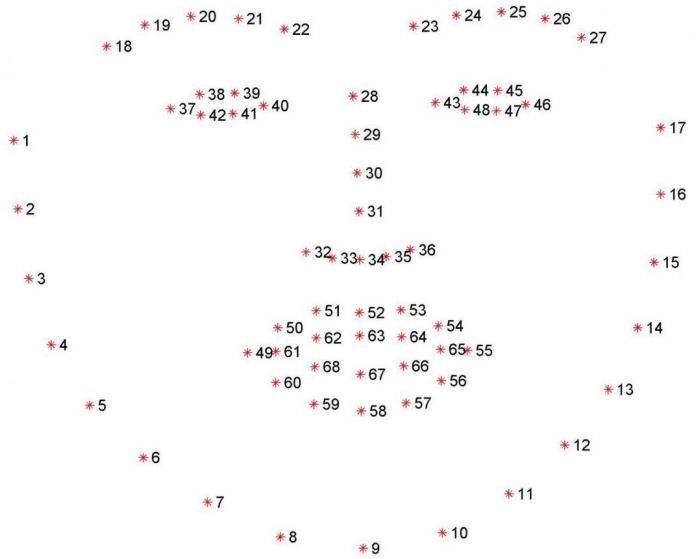
https://github.com/songys/Chatbot_data

송영숙님 데이터 사용

퍼스널 컬러 진단 및
입술 색칠

Dlib 라이브러리를 이용한 얼굴 인식

Face Alignment



- Dlib 라이브러리를 사용하여 detection and alignment 구현
- Face landmark detector는 얼굴에서 68개의 특징점을 찾아 좌표 값으로 리턴 함

Dlib 라이브러리를 이용한 얼굴 인식

Face Alignment

Dlib 라이브러리를 이용한 얼굴 인식

Face Alignment

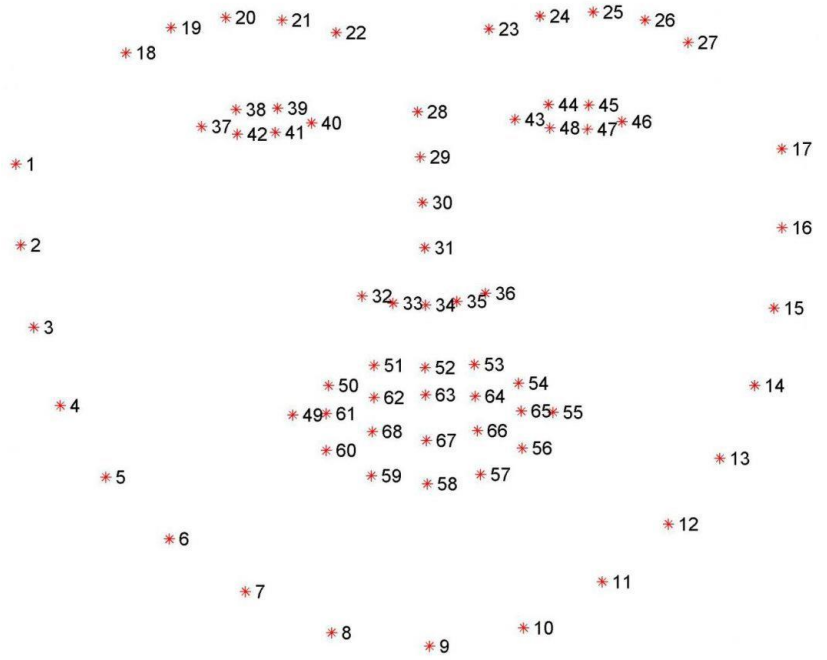
```python
def align_face(filepath, predictor):
    lm = get_landmark(filepath, predictor)

    lm_chin = lm[0: 17]  # left-right
    lm_eyebrow_left = lm[17: 22]  # left-right
    lm_eyebrow_right = lm[22: 27]  # left-right
    lm_nose = lm[27: 31]  # top-down
    lm_nostrils = lm[31: 36]  # top-down
    lm_eye_left = lm[36: 42]  # left-clockwise
    lm_eye_right = lm[42: 48]  # left-clockwise
    lm_mouth_outer = lm[48: 60]  # left-clockwise
    lm_mouth_inner = lm[60: 68]  # left-clockwise

    # Calculate auxiliary vectors.
    eye_left = np.mean(lm_eye_left, axis=0)
    eye_right = np.mean(lm_eye_right, axis=0)
    eye_avg = (eye_left + eye_right) * 0.5
    eye_to_eye = eye_right - eye_left
    mouth_left = lm_mouth_outer[0]
    mouth_right = lm_mouth_outer[6]
    mouth_avg = (mouth_left + mouth_right) * 0.5
    eye_to_mouth = mouth_avg - eye_avg

    # Choose oriented crop rectangle.
    x = eye_to_eye - np.flipud(eye_to_mouth) * [-1, 1]
    x /= np.hypot(*x)
    x *= max(np.hypot(*eye_to_eye) * 2.0, np.hypot(*eye_to_mouth) * 1.8)
    y = np.flipud(x) * [-1, 1]
    c = eye_avg + eye_to_mouth * 0.1
    quad = np.stack([c - x - y, c - x + y, c + x + y, c + x - y])
    qsize = np.hypot(*x) * 2

    # read image
    img = PIL.Image.open(filepath)
```



퍼스널 컬러 진단

- Dlib 라이브러리를 사용하여 얼굴 인식
- 질의 사진의 뺨, 눈동자, 눈썹의 Lab b값을 계산하여

  웜톤 또는 쿨톤으로 분류
- HSV s값을 계산하여

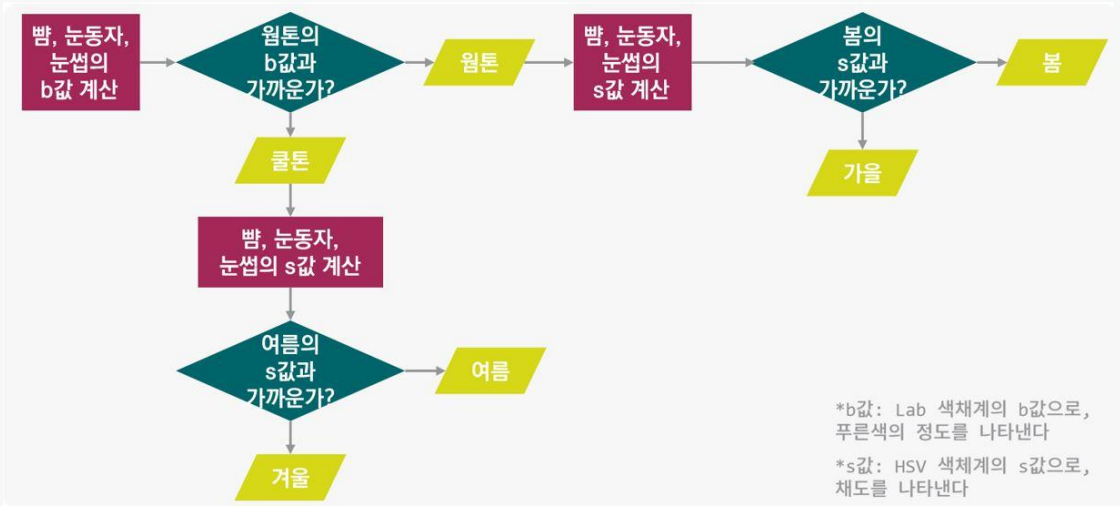
  웜톤이라면 봄 또는 가을로,

  쿨톤이라면 여름 또는 겨울로 분류
- 질의 사진의 색상과 기준 색상의 차이를 구한 후

  피부, 눈썹, 눈동자 별로 설정된 가중치를 곱하여

  합한 값을 비교

  합이 작은 계절이 질의 사진의 퍼스널 컬러로 도출됨




```
eye 의 fall 기준값과의 거리
12.5
./img/test2.jpg의 퍼스널 컬러는 봄웜톤(spring)입니다.
```

```
14.372599999999998
./img/test3.jpg의 퍼스널 컬러는 여름쿨톤(summer)입니다.
```

입술 색칠

- Dlib의 좌표 값 중 입술의 좌표를 속성 값으로 설정
- 보간법 (interpolate)

  1) 스플라인 곡선

     몇 개의 샘플 포인트들로 추정하여

     인접한 점들 사이를 다항식 함수로

     완만한 곡선으로 이어준 것

  2) 미싱 포인트(추정)들을 계산, 추정하는 방법

  3) linear 방식(1차)이 있고,

     2차, 3차, 4차 곡선 등으로 확장 가능

```python
class Lipstick:
    def __init__(self) -> None:
        self.r, self.g, self.b = Red.spring()
        self.up_left_end = 4
        self.up_right_end = 7
        self.in_left_end = 3
        self.in_right_end = 7

        self.lower_left_end = 5
        self.upper_left_end = 11
        self.lower_right_end = 16
        self.upper_right_end = 22

        self.inten = 0.8

        self.x = None
        self.y = None
        self.alpha = None
        self.val = None

        self.im = None
        self.im2 = None

        self.img = np.array(imread("../../img/test1.jpg"))

    @staticmethod
    def inter(lx, ly, k1='quadratic'):
        unew = np.arange(lx[0], lx[-1] + 1, 1)
        f2 = interp1d(lx, ly, kind=k1)
        return f2, unew
```

입술 색칠

주어진 이미지의 입술 좌표 값을 찾아낸 후 사진에 적용

```python
@staticmethod
def getpoint(img):
    predictor_path = "../../data/shape_predictor_68_face_landmarks.dat"
    points = []
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor(predictor_path)
    x = []
    y = []
    dets = detector(img, 1)
    for k, d in enumerate(dets):
        shape = predictor(img, d)
        print("Part 0: {}, Part 1: {} ...".format(shape.part(0),
                                                  shape.part(1)))
        i = 0
        for pt in shape.parts():
            i = i + 1
            x.append(pt.x)
            y.append(pt.y)
        for i in range(0, len(x) - 1):
            if i >= 48 and i <= 59:
                pos = (x[i], y[i])
                points.append(pos)
        pos = (x[48], y[48])
        points.append(pos)
        for i in range(0, len(x) - 1):
            if i >= 60 and i <= 67:
                pos = (x[i], y[i])
                points.append(pos)
                if i == 64:
                    pos = (x[54], y[54])
                    points.append(pos)
        pos = (x[67], y[67])
        points.append(pos)
    return points
```

```python
def apply_getpoint(self):
    inter = self.inter
    up_right_end = self.up_right_end
    in_right_end = self.in_right_end

    img = self.img
    points = self.getpoint(img)
    points=np.array(points)
    point_out_x = np.array((points[:12][:, 0]))
    point_out_y = np.array(points[:12][:, 1])
    point_in_x = np.array(points[12:][:, 0])
    point_in_y = np.array(points[12:][:, 1])

    self.im = img.copy()
    self.im2 = img.copy()


    o_l = self.inter([point_out_x[0]] + point_out_x[up_right_end - 1:][::-1].tolist(),
                [point_out_y[0]] + point_out_y[up_right_end - 1:][::-1].tolist(), 'quadratic')
    o_u = inter(point_out_x[:up_right_end][::-1].tolist(),
                point_out_y[:up_right_end][::-1].tolist(), 'quadratic')

    i_u = inter(point_in_x[:in_right_end][::-1].tolist(),
                point_in_y[:in_right_end][::-1].tolist(), 'quadratic')
    i_l = inter([point_in_x[0]] + point_in_x[in_right_end - 1:][::-1].tolist(),
                [point_in_y[0]] + point_in_y[in_right_end - 1:][::-1].tolist(), 'quadratic')
    self.x = []  # will contain the x coordinates of points on lips
    self.y = []  # will contain the y coordinates of points on lips

    for i in range(int(point_in_x[0]),int(point_in_x[6])):
    #   k = k + 1
        for j in range(int(o_u[0](i)),int(i_u[0](i))):
            self.x.append(j)
            self.y.append(i)
        for j in range(int(i_l[0](i)), int(o_l[0](i))):
            self.x.append(j)
            self.y.append(i)
```

```python
for i in range(int(point_out_x[0]),int(point_in_x[0])):
    for j in range(int(o_u[0](i)),int(o_l[0](i))):
        self.x.append(j)
        self.y.append(i)

for i in range(int(point_in_x[6]),int(point_out_x[6])):
    for j in range(int(o_u[0](i)),int(o_l[0](i))):
        self.x.append(j)
        self.y.append(i)
```

입술 색칠

- 이미지의 고주파 부분(노이즈)를 자연스럽게 바꿔주기 위해 블러링함
- 가우시안 블러링

  1) 가운데 픽셀에 가장 큰 가중치를 두어 계산

  2) 가우시안 분포(정규 분포)를 갖는 커널로 블러링 수행

  3) 정규 분포

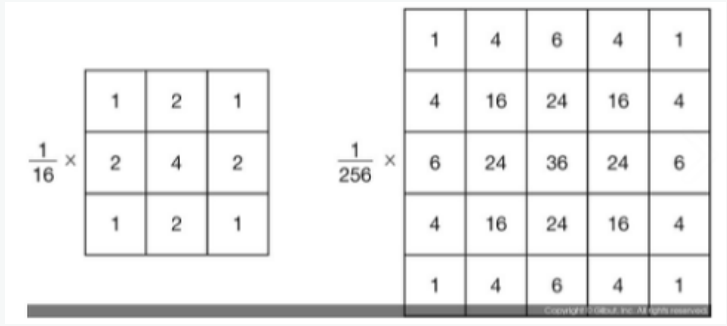
  평균 근처에 몰려 있는 값들의 개수가 많고 평균에서 멀어질수록 그 개수가 적어지는 분포

```python
def guassian_blur(self):
    # guassian blur
    height,width = self.im.shape[:2]
    filter = np.zeros((height,width))
    cv2.fillConvexPoly(filter,np.array(c_[self.y, self.x],dtype = 'int32'),1)

    filter = cv2.GaussianBlur(filter,(31,31),0)

    # Erosion to reduce blur size
    kernel = np.ones((10,10),np.uint8)
    filter = cv2.erode(filter,kernel,iterations = 1)
    self.alpha=np.zeros([height,width,3],dtype='float64')
    self.alpha[:,:,0]=filter
    self.alpha[:,:,1]=filter
    self.alpha[:,:,2]=filter

def save(self):
    imsave('../../save/virtual_makeup/lips/test1.jpg', (self.alpha*self.im+(1-self.alpha)*self.im2).astype('uint8'))
```



[그림 1] 가우시안 필터

입술 색칠 결과

before

after

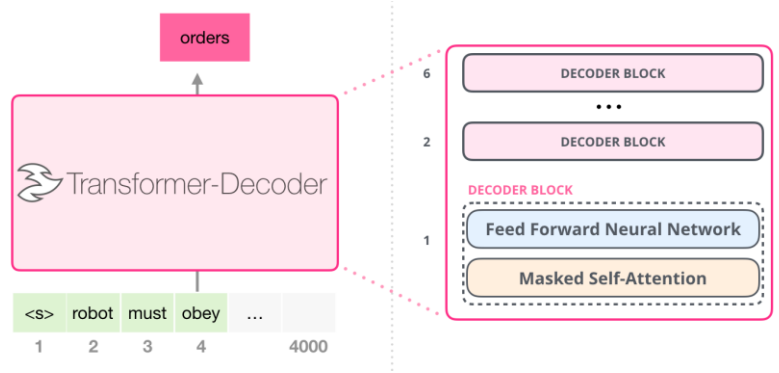
before

after

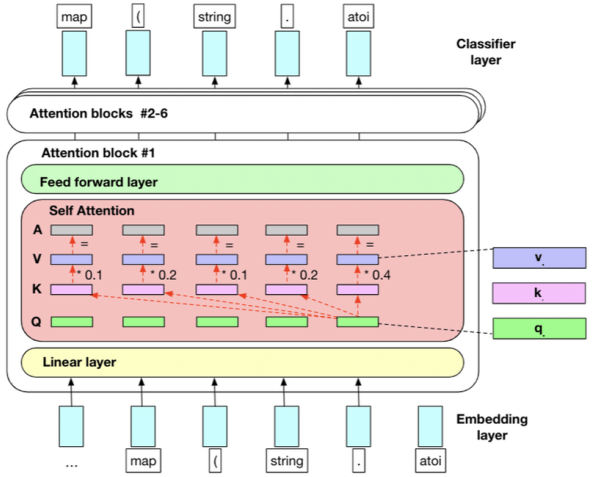
챗봇

GPT-2

- Transformer 구조에서 Decoder 부분만 사용
- 단방향 언어 모델
- 문장 생성에 강점



[그림 2] Transformer-Decoder 모델 구조



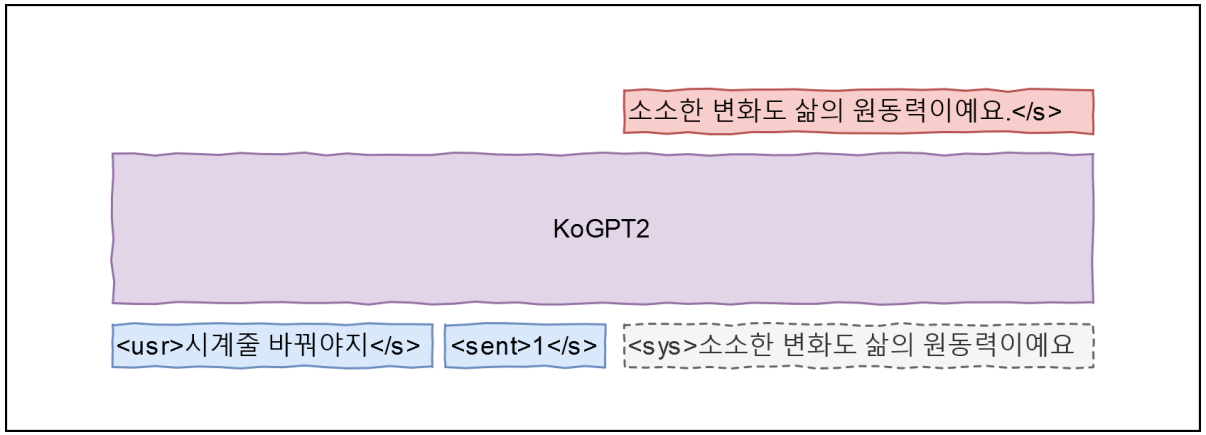
[그림 3] GPT-2 Transformer 모델 구조

KoGPT2

- 한국어로 학습된 오픈소스 기반 GPT-2 모델
- 공개된 한글 챗봇 데이터와

  pretrained KoGPT2를 이용해

  간단한 대화가 가능한 챗봇 구현



KoGPT2

KoGPT2 모델을 사용하기 위해 필요한 라이브러리 설치

```
import urllib
import pandas as pd
import torch
from torch.utils.data import DataLoader
from transformers import PreTrainedTokenizerFast, GPT2LMHeadModel
from tqdm import tqdm
from chatbot_dataset import ChatbotDataset


class Kogpt:
    def __init__(self):
        self.Q_TKN = "<usr>"
        self.A_TKN = "<sys>"
        self.BOS = '</s>'
        self.EOS = '</s>'
        self.MASK = '<unused0>'
        self.SENT = '<unused1>'
        self.PAD = '<pad>'
        self.epoch = 30
        self.Sneg = -1e18
        self.log_interval = 200
        self.device = torch.device('cuda:0')

    def hook(self):
        self.train()
        self.save_model()

    def crawling(self):
        urllib.request.urlretrieve(
            "https://raw.githubusercontent.com/songys/Chatbot_data/master/ChatbotData.csv",
            filename="data/ChatBotData.csv")

    def load_tokenizer(self):
        koGPT2_TOKENIZER = PreTrainedTokenizerFast.from_pretrained("skt/kogpt2-base-v2", bos_token=self.BOS,
```

KoGPT2를 이용한 전이학습과 파인튜닝

```
                                                    eos_token=self.EOS, unk_token="<unk>",
                                                    pad_token=self.PAD, mask_token=self.MASK)
        return koGPT2_TOKENIZER

    def load_model(self):
        model = GPT2LMHeadModel.from_pretrained('skt/kogpt2-base-v2').to(self.device)
        return model

    @staticmethod
    def collate_batch(batch):
        data = [item[0] for item in batch]
        mask = [item[1] for item in batch]
        label = [item[2] for item in batch]
        return torch.LongTensor(data), torch.LongTensor(mask), torch.LongTensor(label)

    def train(self):
        Chatbot_Data = pd.read_csv("./data/ChatBotData.csv")
        train_set = ChatbotDataset(Chatbot_Data, max_len=40)

        # 윈도우 환경에서 num_workers 는 무조건 0으로 지정, 리눅스에서는 2
        train_dataloader = DataLoader(train_set, batch_size=32, num_workers=0, shuffle=True,
                                      collate_fn=self.collate_batch)

        model = self.load_model()
        # model.to(device)
        model.train()

        learning_rate = 3e-5
        criterion = torch.nn.CrossEntropyLoss(reduction="none")
        optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
        print("start")
        for epoch in range(self.epoch):
            train_acc = 0.0
            for batch_idx, samples in enumerate(tqdm(train_dataloader)):
```

```
                optimizer.zero_grad()
                token_ids, mask, label = samples
                token_ids = token_ids.long().to(self.device)
                mask = mask.long().to(self.device)
                label = label.long().to(self.device)
                out = model(token_ids)
                out = out.logits  # Returns a new tensor with the logit of the elements of input
                mask_3d = mask.unsqueeze(dim=2).repeat_interleave(repeats=out.shape[2], dim=2)
                mask_out = torch.where(mask_3d == 1, out, self.Sneg * torch.ones_like(out))
                loss = criterion(mask_out.transpose(2, 1), label)
                # 평균 loss 만들기 avg_loss[0] / avg_loss[1] <- loss 정규화
                avg_loss = loss.sum() / mask.sum()
                avg_loss.backward()
                # 학습 끝
                optimizer.step()
                if batch_idx % self.log_interval == 0:
                    print("epoch {} batch id {}0: loss {} ".format(epoch+1, batch_idx+1, avg_loss))
        print("end")
        return model

    def save_model(self):
        model = self.train()
        PATH = './model/'
        torch.save(model, PATH + 'model.pt')


if __name__ == '__main__':
    Kogpt().hook()
```

```
epoch 1 batch id 10: loss 34.6140022277832
 54%|█████     | 201/370 [00:44<00:37,  4.52it/s]epoch 1 batch id 2010: loss 35.408260345458984
100%|██████████| 370/370 [01:21<00:00,  4.55it/s]
  0%|          | 1/370 [00:00<01:47,  3.42it/s]epoch 2 batch id 10: loss 33.623146057128906
 54%|█████     | 201/370 [00:43<00:37,  4.45it/s]epoch 2 batch id 2010: loss 31.20586585998535
100%|██████████| 370/370 [01:20<00:00,  4.58it/s]
  0%|          | 1/370 [00:00<01:48,  3.40it/s]epoch 3 batch id 10: loss 32.3988151550293
 54%|█████     | 201/370 [00:44<00:38,  4.44it/s]epoch 3 batch id 2010: loss 33.265193939208984
100%|██████████| 370/370 [01:21<00:00,  4.57it/s]
  0%|          | 1/370 [00:00<01:49,  3.37it/s]epoch 4 batch id 10: loss 30.95209312438965
 54%|█████     | 201/370 [00:44<00:38,  4.42it/s]epoch 4 batch id 2010: loss 34.23511505126953
100%|██████████| 370/370 [01:21<00:00,  4.54it/s]
  0%|          | 1/370 [00:00<01:48,  3.39it/s]epoch 5 batch id 10: loss 29.72269058227539
 16%|█         | 59/370 [00:13<01:08,  4.53it/s]
```

데이터 크롤링

퍼스널 컬러 진단 사이트인 colorize의 Beauty Patch 커뮤니티에서
퍼스널 컬러 별 화장품 추천 게시글 크롤링

```
class Cosmetics():
    def __init__(self) -> None:
        pass

    def recommend_cosmetics(self):
        item_info = []
        for page in range(35):
            url = 'https://www.colorize.co.kr/shop/board/list.php?id=beautypatch&page=' + str(page)
            html_doc = urlopen(url)
            soup = BeautifulSoup(html_doc, 'lxml')
            info = soup.find_all('td', {'class':'nwz-info'})

            for i in info:
                i = i.get_text().split(sep='\r\n\t\t\t\t')
                item_info.append([j.strip() for j in i])

        for i in item_info:
            del i[-1]

        df = pd.DataFrame(item_info, columns=['퍼스널컬러', '브랜드', '제품명'])
        df.dropna(axis=0, inplace=True)
        df.to_csv('../save/recommend_cosmetics.csv', encoding='utf-8')


if __name__ == '__main__':
    Cosmetics().recommend_cosmetics()
```

데이터 분류

수집한 데이터를 퍼스널 컬러 종류별로 분류한 후
그 중 하나의 데이터를 랜덤으로 반환

```python
class Recommend:
    def __init__(self) -> None:
        self.data = pd.read_csv('./save/recommend_cosmetics.csv', encoding='utf-8')

    def spring(self):
        return self.data.loc[self.data['퍼스널컬러']=='봄웜'].iloc[:, 2:4].sample()

    def summer(self):
        return self.data.loc[self.data['퍼스널컬러']=='여름쿨'].iloc[:, 2:4].sample()

    def fall(self):
        return self.data.loc[self.data['퍼스널컬러']=='가을웜'].iloc[:, 2:4].sample()

    def winter(self):
        return self.data.loc[self.data['퍼스널컬러']=='겨울쿨'].iloc[:, 2:4].sample()
```

챗봇 실행 코드 및 결과

```python
import torch
from transformers import PreTrainedTokenizerFast
from recommend_cosmetics import Recommend


class Chatbot:
    def __init__(self) -> None:
        self.Q_TKN = "<usr>"
        self.A_TKN = "<sys>"
        self.BOS = '</s>'
        self.EOS = '</s>'
        self.MASK = '<unused0>'
        self.SENT = '<unused1>'
        self.PAD = '<pad>'
        self.model_path = './model'

    def chatbot(self):
        Q_TKN = self.Q_TKN
        SENT = self.SENT
        A_TKN = self.A_TKN
        BOS = self.BOS
        EOS = self.BOS
        PAD = self.PAD
        MASK = self.MASK
        device = torch.device("cpu")
        koGPT2_TOKENIZER = PreTrainedTokenizerFast.from_pretrained("skt/kogpt2-base-v2",
                                                   bos_token=BOS, eos_token=EOS, unk_token="<unk>", pad_token=PAD, mask_token=MASK)
        model = torch.load(f'{self.model_path}/model.pt', map_location=device)
        r = Recommend()
        with torch.no_grad():
            intro = "안녕하세요"
            print(f"Chatbot > {intro}")
            while 1:
                q = input("user > ").strip()
                if q == "종료":
                    break
                elif '화장품' in q:
                    if '봄' in q:
                        print(f'{r.spring()} 을(를) 추천해드려요.')
                        continue
                    elif '여름' in q:
                        print(f'{r.summer()} 을(를) 추천해드려요.')
                        continue
                    elif '가을' in q:
                        print(f'{r.fall()} 을(를) 추천해드려요.')
                        continue
                    elif '겨울' in q:
                        print(f'{r.winter()} 을(를) 추천해드려요.')
                        continue
                a = ""
                while 1:
                    input_ids = torch.LongTensor(koGPT2_TOKENIZER.encode(Q_TKN + q + SENT + A_TKN + a)).unsqueeze(dim=0)
                    pred = model(input_ids)
                    pred = pred.logits
                    gen = koGPT2_TOKENIZER.convert_ids_to_tokens(torch.argmax(pred, dim=-1)
                                                                 .squeeze().numpy().tolist())[-1]
                    if gen == EOS:
                        break
                    a += gen.replace("_", " ")
                print("Chatbot > {}".format(a.strip()))


if __name__ == '__main__':
    Chatbot().chatbot()
```

```
Chatbot > 안녕하세요
user > 안녕
Chatbot > 안녕하세요
user > 봄웜톤 화장품 추천해줘
    브랜드             제품명
367  디올  어딕트 립 글로우 코랄 을(를) 추천해드려요.
user > 가을웜 화장품에는 뭐가 있어?
      브랜드             제품명
508  슈에무라  강남오렌지 OR570 을(를) 추천해드려요.
user > 여름쿨톤 화장품 추천해주세요
     브랜드         제품명
55  입생로랑  입생로랑 11호 을(를) 추천해드려요.
user > 겨울쿨 화장품 알려줘
     브랜드       제품명
397  키스미  블랙마스카라 을(를) 추천해드려요.
```

화면 기술 스택

화면 기술 스택

협업을 위해 간단하게 구현

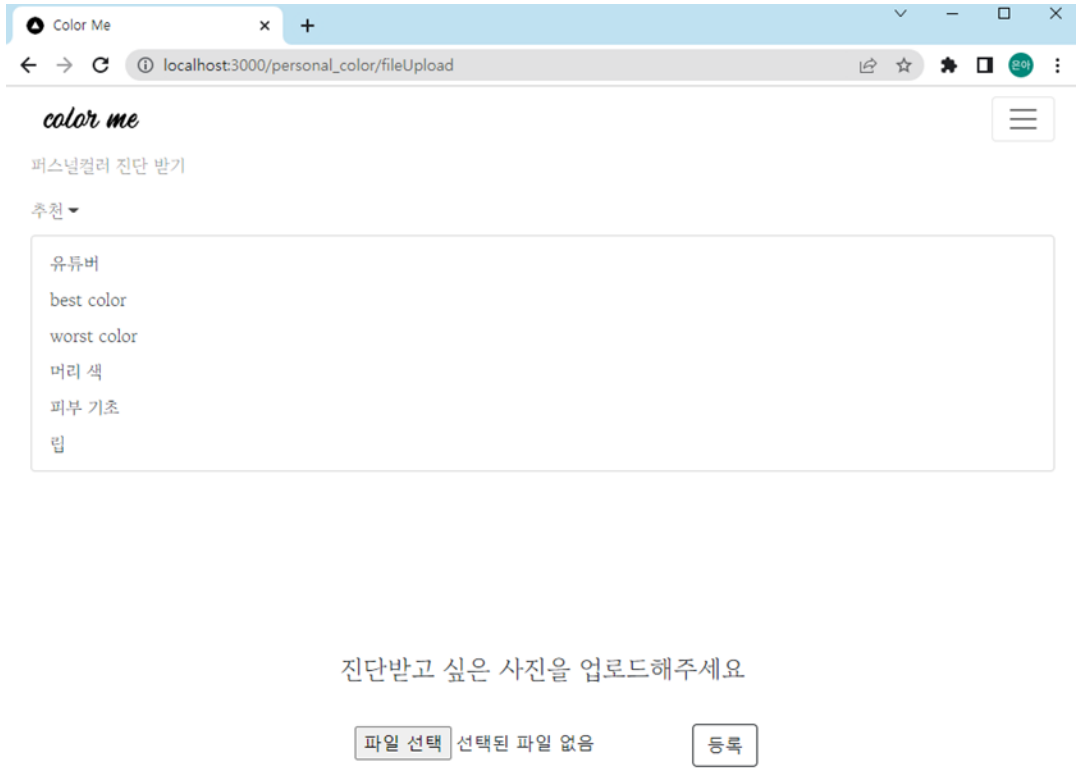
NEXT.JS

React

TS TypeScript

Redux Toolkit

Redux-Saga



Q&A