# 형태소 기반 폰트 추천 시스템

## 디렉터리 구조

📁 img2vec_pytorch

📁 font_recommendation

- 📁 bbichim
- 📁 buri
- 📁 kkeokim
- 📁 kkokjijum
- 📁 sangtu
- 📁 test_img
- 📄 font_recommendation_based_on_stroke_elements.ipynb

## 클러스터링

### part1. 차원축소

```
def find_cluster(df, centroid, pic, pca_n, candidate_font, comparative_font_df):

  # df엔 cluster 결과가 포함되어있으므로, pca를 하기위해 cluster 열을 제외함
  # (pca할 땐 특징 벡터값만 있어야한다)
  prev_pca = df.drop(columns=['cluster'])

  # 입력 형태소 이미지를 df에 포함해 같이 차원축소한다.
  prev_pca.loc[len(df)] = pic

  # 입력 형태소 이미지를 포함해 전체 데이터 차원축소
  pca = data_pca(prev_pca, pca_n)
```

### part2. 클러스터링

```
  # 입력 이미지의 특징벡터
  input = pca.iloc[[-1]]

  # 입력이미지의 벡터값과 각 클러스터별 중심점의 벡터값의 거리를 코사인유사도로 비교
  # -> 입력 이미지가 어느 클러스터에 속하는지 계산한다
  dist_to_centroid = []
  for idx, row in centroid.iterrows():
    dist_to_centroid.append(cosine_similarity(input.values, centroid.loc[[idx]].values)[0][0])
  input_cluster = dist_to_centroid.index(max(dist_to_centroid))
  print("input_cluster : ", input_cluster)

  # 입력이미지와 위의 클러스터에 속한 형태소 이미지의 유사도 계산
  similarity = []
  same_cluster_idx = df[df['cluster']==input_cluster].index
  for idx in same_cluster_idx:
    similarity.append(cosine_similarity(input.values, pca.iloc[[idx]].values)[0][0])

  # 유사도가 0.6보다 큰 폰트만 후보 폰트(최종 유사도 계산 대상 폰트)로 함
  similarity_df = add_fontname(pca).loc[same_cluster_idx, :]
  similarity_df['similarity'] = similarity
  sim_high_df = similarity_df[similarity_df['similarity'] >= 0.6]
  candidate_font = np.concatenate((candidate_font, sim_high_df['fontname'].values), axis=0)

  # 클러스터링이 끝난 후 입력 형태소와 기존 형태소의 비교를 위해 기존 폰트의 형태소를 df에 합침
  comparative_font_df = pd.concat([comparative_font_df, pca.drop(columns=['fontname'])], axis=1)
  return candidate_font, comparative_font_df
```

## candidate_font와 comparative_font_df

### candidate_font

입력 형태소 이미지와 비교할 폰트의 목록(list)이다. 입력 형태소 이미지가 속한 클러스터에서 입력 형태소 이미지와 비교했을 때 유사도가 0.6 이상인 형태소 이미지의 폰트 명을 저장한다.

### comparative_font_df

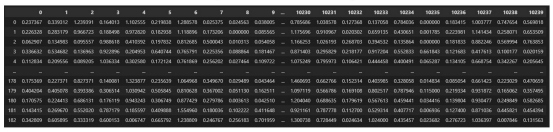
형태소의 특징벡터 df를 concatenate한다. 입력 이미지로부터 항상 모든 형태소가 검출되는 것은 아니기 때문에 검출된 형태소의 df만 concatenate한다. 예를 들어 삐침과 상투가 검출됐으면 삐침 df와 상투 df만 concatenate한다. 이렇게 만들어진 comparative_font_df는 입력 형태소의 특징벡터와 코사인 유사도로 비교(유사도를 계산)한다. 계산이 끝나면 폰트 추천 결과를 얻을 수 있다.

```
# 5. 입력 형태소 이미지와 비교 대상 폰트의 형태소 이미지를 비교
comparative_font_df = add_fontname(comparative_font_df)
test_font_vec = comparative_font_df.iloc[[-1]].iloc[:, 1:]  # 입력 이미지의 형태소 특징벡터

font_recommendation_list = []
font_recommendation_sim = []

for i in range(comparative_font_df.shape[0]):
  if comparative_font_df.iloc[i]['fontname'] in candidate_font:
    font_recommendation_list.append(comparative_font_df.iloc[i]['fontname'])
    font_recommendation_sim.append(cosine_similarity(test_font_vec.values, comparative_font_df.iloc[[i]].iloc[:, 1:].values)[0][0])
```

1. 획요소별 특징 벡터 추출 (2048 columns)

<삐침 특징 벡터> <부리 특징 벡터> <꺾임 특징 벡터> <상투 특징 벡터> <꼭지점 특징 벡터>

2. 각 특징 벡터를 concat(10240 column)



<삐침+부리+꺾임+상투+꼭지점 특징 벡터>

다섯가지 형태소의 특징벡터를 모두 concat한 데이터프레임