CH6 Kaggle

차원축소 기법

https://www.kaggle.com/code/shivamb/dataset-decomposition-techniques

▼ 필사

```
# Dataset Preparation
train = pd.read_csv(path+'train.csv')
target = train['target']
train = train.drop(["target", "ID"], axis=1)
print ("Rows: " + str(train.shape[0]) + ", Columns: " + str(train.shape[1]))
train.head()
standardized_train = StandardScaler().fit_transform(train.values)

#Feature Statistics
feature_df = train.describe().T
feature_df = feature_df.reset_index().rename(columns = {'index' : 'columns'})
feature_df['distinct_vals'] = feature_df['columns'].apply(lambda x : len(train[x].value_counts()))
feature_df['column_var'] = feature_df['columns'].apply(lambda x : np.var(train[x]))
feature_df['column_std'] = feature_df['columns'].apply(lambda x : np.std(train[x]))
feature_df['column_mean'] = feature_df['columns'].apply(lambda x : np.mean(train[x]))
feature_df['target_corr'] = feature_df['columns'].apply(lambda x : np.corrcoef(target, train[x])[0][1])
feature_df.head()

feature_df = feature_df.sort_values('column_var', ascending = True)
feature_df['column_var'] = (feature_df['column_var'] - feature_df['column_var'].min()) / (feature_df['column_var'].max() - feature
trace1 = go.Scatter(x=feature_df['columns'], y=feature_df['column_var'], opacity=0.75, marker=dict(color="red"))
layout = dict(height=400, title='Feature Variance', legend=dict(orientation="h"));
fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);

trace1 = go.Histogram(x=feature_df[feature_df['column_var'] <= 0.01]['column_var'], opacity=0.45, marker=dict(color="red"))
layout = dict(height=400, title='Distribution of Variable Variance <= 0.01', legend=dict(orientation="h"));
fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);

trace1 = go.Histogram(x=feature_df[feature_df['column_var'] > 0.01]['column_var'], opacity=0.45, marker=dict(color="red"))
layout = dict(height=400, title='Distribution of Variable Variance > 0.01', legend=dict(orientation="h"));
fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);

trace1 = go.Histogram(x=feature_df['target_corr'], opacity=0.45, marker=dict(color="green"))
layout = dict(height=400, title='Distribution of correlation with target', legend=dict(orientation="h"));
fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);

# Decomposition into EigenVectors and EigenValues

# Calculating Eigenvectors and eigenvalues of Cov matirx
mean_vec = np.mean(standardized_train, axis=0)
cov_matrix = np.cov(standardized_train.T)
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)

# Create a list of (eigenvalue, eigenvector) tuples
eig_pairs = [ (np.abs(eig_vals[i]),eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the eigenvalue, eigenvector pair from high to low
eig_pairs.sort(key = lambda x: x[0], reverse= True)

# Calculation of Explained Variance from the eigenvalues
tot = sum(eig_vals)

# Individual explained variance
var_exp = [(i/tot)*100 for i in sorted(eig_vals, reverse=True)]
var_exp_real = [v.real for v in var_exp]

# Cumulative explained variance
cum_var_exp = np.cumsum(var_exp)
cum_exp_real = [v.real for v in cum_var_exp]

## plot the variance and cumulative variance
trace1 = go.Scatter(x=train.columns, y=var_exp_real, name="Individual Variance", opacity=0.75, marker=dict(color="red"))
trace2 = go.Scatter(x=train.columns, y=cum_exp_real, name="Cumulative Variance", opacity=0.75, marker=dict(color="blue"))
layout = dict(height=400, title='Variance Explained by Variables', legend=dict(orientation="h", x=0, y=1.2));
```

```
fig = go.Figure(data=[trace1, trace2], layout=layout);
iplot(fig);
```

1. PCA 변형기법

- **Kernel PCA**
  - **커널을 사용하여 비선형 차원 축소 방법**
  - 커널을 이용해 데이터를 저차원에서 고차원으로 매핑시켜 비선형 데이터셋에 SVM을 적용시키는 Kernel SVM을 PCA에 적용하여 비선형 투영으로 차원 축소
  - 노이즈 제거, compression and structured prediction에 사용

```
from sklearn.datasets import make_swiss_roll

X, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)
from sklearn.decomposition import KernelPCA

rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
lin_pca = KernelPCA(n_components = 2, kernel="linear", fit_inverse_transform=True)
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.0433, fit_inverse_transform=True)
sig_pca = KernelPCA(n_components = 2, kernel="sigmoid", gamma=0.001, coef0=1, fit_inverse_transform=True)

y = t > 6.9

plt.figure(figsize=(11, 4))
for subplot, pca, title in ((131, lin_pca, "선형 커널"), (132, rbf_pca, "RBF 커널, $\gamma=0.04$"), (133, sig_pca, "시그모이드 커널, $\g
   X_reduced = pca.fit_transform(X)
   if subplot == 132:
       X_reduced_rbf = X_reduced

   plt.subplot(subplot)
   #plt.plot(X_reduced[y, 0], X_reduced[y, 1], "gs")
   #plt.plot(X_reduced[~y, 0], X_reduced[~y, 1], "y^")
   plt.title(title, fontsize=14)
   plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=t, cmap=plt.cm.hot)
   plt.xlabel("$z_1$", fontsize=18)
   if subplot == 131:
       plt.ylabel("$z_2$", fontsize=18, rotation=0)
   plt.grid(True)

plt.show()
```

- **Incremental PCA**
  - SVD 수행을 위해 전체 학습 데이터셋을 메모리에 올려야 한다는 PCA의 단점을 보완하기 위해 등장
  - 학습 데이터셋을 미니배치로 나눈 뒤 IPCA 알고리즘에 의해 하나의 미니배치를 입력으로 넣어준다.
  - 학습 데이터셋 클 때 좋음
  - memory efficient

```
from sklearn.decomposition import IncrementalPCA

n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for batch_x in np.array_split(train_x, n_batches):
    print(".", end="") # not shown in the book
    inc_pca.partial_fit(batch_x)

X_reduced = inc_pca.transform(train_x)
X_recovered_inc_pca = inc_pca.inverse_transform(X_reduced)
plt.figure(figsize=(7, 4))
plt.subplot(121)
plot_digits(train_x[::2100])
plt.subplot(122)
plot_digits(X_recovered_inc_pca[::2100])
plt.tight_layout()
```

- **Sparse PCA**
  - 피쳐가 매우 많은 자료에 대해서 사용할 수 있는 PCA기법
  - 전체 피쳐를 사용하는 것이 아니라, 중요하다고 생각되는 피쳐에 주목

- 조각 영상을 여러 개의 기저 벡터 선형 결합으로 표현

  조각영상을 x, i번째 기저 벡터를 d_i, 계수집합을 a로 표기하면, 희소코딩 과정은 다음과 같습니다.

  $$\{ x = Da \ where, D = (d_1 d_2 \cdots d_m)$$

  d는 조각 영상의 화소 개수이자 벡터 x의 차원 수입니다. 희소 코딩에서는 D를 사전(dictionary), d_i를 사전 요소, 벡터 a는 희소코드(sparse code)라고 합니다.

- 기저함수는 비지도 학습이 훈련집합 X를 통해 알아냄
- 최적의 사전과 최적의 희소코드를 알아내는 것이 목표

- **Mini Batch Sparse PCA**
  - Sparse PCA에 미니 배치 적용
  - faster, less accurate

2. ICA
   - Independent Component Analysis
   - 가장 독립적인 축 찾기 → 독립성이 최대가 되도록
   - **독립성과 비가우시안 가정**(신호 분리 가능) 만족해야 함
   - 밀도 추정, 특징 추출, 신호 분리 등에 사용
   - 학습 : 훈련 집합의 평균이 0벡터가 되도록 이동변환

$$x_i' = (D^{-\frac{1}{2}} V^T) x_i$$

→ Whitening 변환 전처리

→ wj 초기화 → 첨도가 작아지는 방향으로 zj가 비가우시안인 정도를 최대화하는 wj 추정

$$kurtosis(\mathbb{z}_j) = \frac{1}{n}\sum_{i=1}^{n} z_{ji}^4 - 3\left(\frac{1}{n}\sum_{i=1}^{n} z_{ji}^2\right)^2$$

** 첨도는 4차 모멘트까지 사용

```
### Implement ICA
obj_ica = FastICA(n_components = 30)
X_ica = obj_ica.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_ica, 'ICA - First three components')
plot_2_components(X_ica, 'ICA - First two components')
```

3. factor analysis

- 요인 분석 → 관측된 변수 집합에서 영향력 있는 기본 요인 또는 잠재 요인을 검색하는 데 사용되는 탐색적 데이터 분석 방법
- A simple linear generative model with Gaussian latent variables
- 요인 = 관측된 변수의 수 사이의 연관성을 설명하는 잠재 변수
- 분산의 양이 가장 적은 요인은 제거됨
- indicator의 원인 → factor

PCA	요인 분석
축 직교	요인 직교할 필요 없음
관측된 변수의 선형 조합	요인 분석에서 관측된 변수는 관측되지 않은 변수/요인의 선형 조합
구성 요소 해석 불가	기본 요인 레이블링/해석 가능
차원 축소 기법 (요인 분석 중 하나)	잠재변수 방법
최대 분산 양 설명	공분산 설명

```
### Implement Factor Analysis
obj_fa = FactorAnalysis(n_components = 30)
X_fa = obj_fa.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_fa, 'Factor Analysis - First three components')
# plot_2_components(X, 'Factor Analysis - First two components')
```

4. t-SNE

- 비선형 관계를 이용한 데이터셋 분해 방법
- t-Distributed Stochastic Neighbor Embedding: 고차원 데이터 탐색에 사용되는 비선형 차원 축소 알고리즘
- 고차원 공간에서 점세트를 가져와 저차원 공간에서 해당 점의 표현을 찾는 것이 목표
- 원본 특성 공간에서 가까운 포인트는 가깝게, 멀리 떨어진 포인트는 멀어지게 표현하여 원래의 데이터 포인트 사이 거리를 잘 보존하는 2차원 표현을 찾는 것이다.

```
tsne_model = TSNE(n_components=2, verbose=1, random_state=42, n_iter=500)
tsne_results = tsne_model.fit_transform(X_svd)

traceTSNE = go.Scatter(
    x = tsne_results[:,0],
    y = tsne_results[:,1],
    name = target,
     hoveron = target,
    mode = 'markers',
    text = target,
    showlegend = True,
    marker = dict(
        size = 8,
        color = '#c94ff2',
        showscale = False,
        line = dict(
            width = 2,
            color = 'rgb(255, 255, 255)'
        ),
        opacity = 0.8
    )
)
data = [traceTSNE]

layout = dict(title = 'TSNE (T-Distributed Stochastic Neighbour Embedding)',
              hovermode= 'closest',
              yaxis = dict(zeroline = False),
              xaxis = dict(zeroline = False),
              showlegend= False)

fig = dict(data=data, layout=layout)
iplot(fig)
```

이미지 데이터 차원축소 mnist 예제

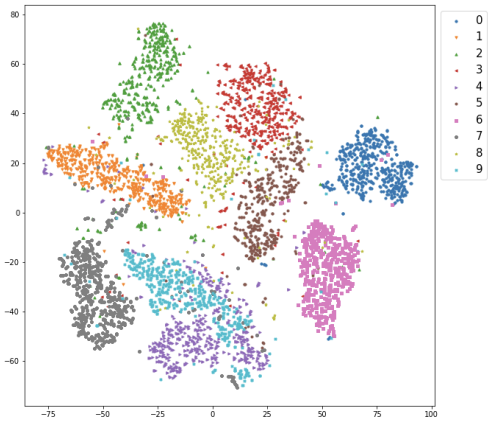
https://www.kaggle.com/code/ohseokkim/the-curse-of-dimensionality-dimension-reduction

[코드]

```
from keras.datasets import mnist
(train_x, train_y), (test_x, test_y) = mnist.load_data()

# Reshape to 2D data
train_x = train_x.reshape(train_x.shape[0], -1)
print(train_x.shape)

sample_size = 5000
# Use only the top 1000 data for training
```

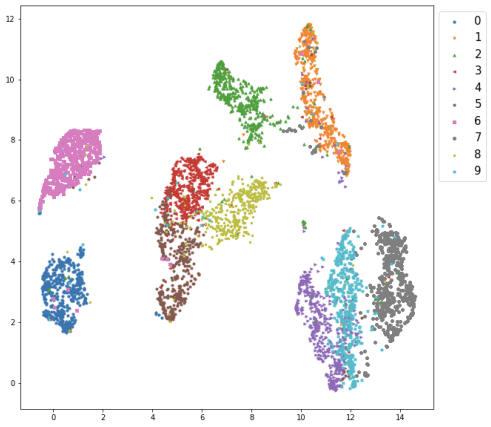
```
train_x = pd.DataFrame(train_x[:sample_size, :])
train_y = train_y[:sample_size]
########################################
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
x_tsne = tsne.fit_transform(train_x)

plt.figure(figsize=(10,10))
for i,marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_tsne[mask, 0], x_tsne[mask, 1], label=i, s=10,
    alpha=1,marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left',fontsize=15)
########################################
import umap
um = umap.UMAP()
x_umap = um.fit_transform(train_x)

plt.figure(figsize=(10,10))
for i,marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_umap[mask, 0], x_umap[mask, 1], label=i, s=10,
    alpha=1,marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left',fontsize=15)

#UMAP connectivity plot
import umap.plot
mapper = umap.UMAP().fit(train_x)
umap.plot.connectivity(mapper, show_points=True)

#UMAP 3D plot
import plotly
import plotly.express as px
from umap import UMAP

umap_3d = UMAP(n_components=3, init='random', random_state=0)
x_umap = umap_3d.fit_transform(train_x)
umap_df = pd.DataFrame(x_umap)
train_y_sr = pd.Series(train_y,name='label')
print(type(x_umap))
new_df = pd.concat([umap_df,train_y_sr],axis=1)
fig = px.scatter_3d(
    new_df, x=0, y=1, z=2,
    color='label', labels={'color': 'number'}
)
fig.update_traces(marker_size=1)
fig.show()
```

1. t-SNE



- 데이터를 2차원 평면으로 압축하여 시각화하는데 자주 사용
- 비선형 관계를 식별할 수 있으므로 압축 결과를 원래 특성에 추가하여 성능 향상
- 비용이 높기 때문에 3차원 이상 압축에는 부적합

2. UMAP



- Uniform Manifold Approximation and Projection
- 비선형 차원 축소를 위해 t-SNE보다 빠르고 데이터 공간을 잘 분리하는 방법
- 매우 큰 데이터 셋 빠르게 처리
- 희소 행렬 데이터에도 적합

[How UMAP Works]

https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

- umap인 t-SNE와는 달리 euclidean distance를 꼭 쓰지 않고 고차원에서 지수 확률 분포를 사용하고, 확률도 normalize 필요 x

- UMAP uses the **number of nearest neighbors** instead of perplexity. While tSNE defined perplexity according to Eq. (2), UMAP defines the number of nearest neighbor **k** without the log2 function, i.e. as follows:

$$k = 2^{\sum_i p_{ij}}$$

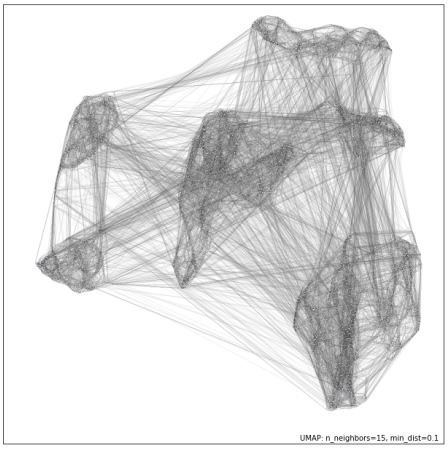
UMAP은 Perplexity 대신에 근처 이웃의 수를 사용한다. UMAP은 근처 이웃의 수를 log2 function 없이 k개의 이웃의 수로 정의한다.

- UMAP uses a **slightly different symmetrization** of the high-dimensional probability

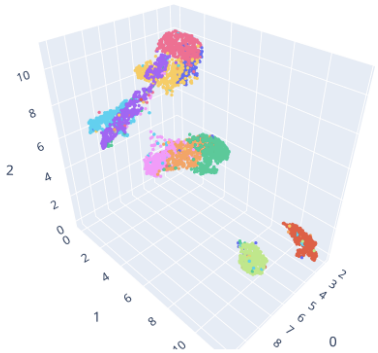
$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$$

- t-SNE의 KL-divergence 대신 비용함수로 binary cross-entropy 사용
  → **Global Structure 보장,,,,,인데 내용이 많이 어렵다,,,,**

3. UMAP connectivity plot



4. UMAP 3D plot



- 2차원보다 복잡하고 sparse하게 분포, 차원이 커질수록 심해진다.