Document Classification and Clustering Using Python

406.426B 데이터관리와 분석

2023.11.25

강명오

mo970610@snu.ac.kr

서울대학교 산업공학과

Document Classification and Clustering Using Python

PART 1. CLASSIFICATION

Classification tutorial

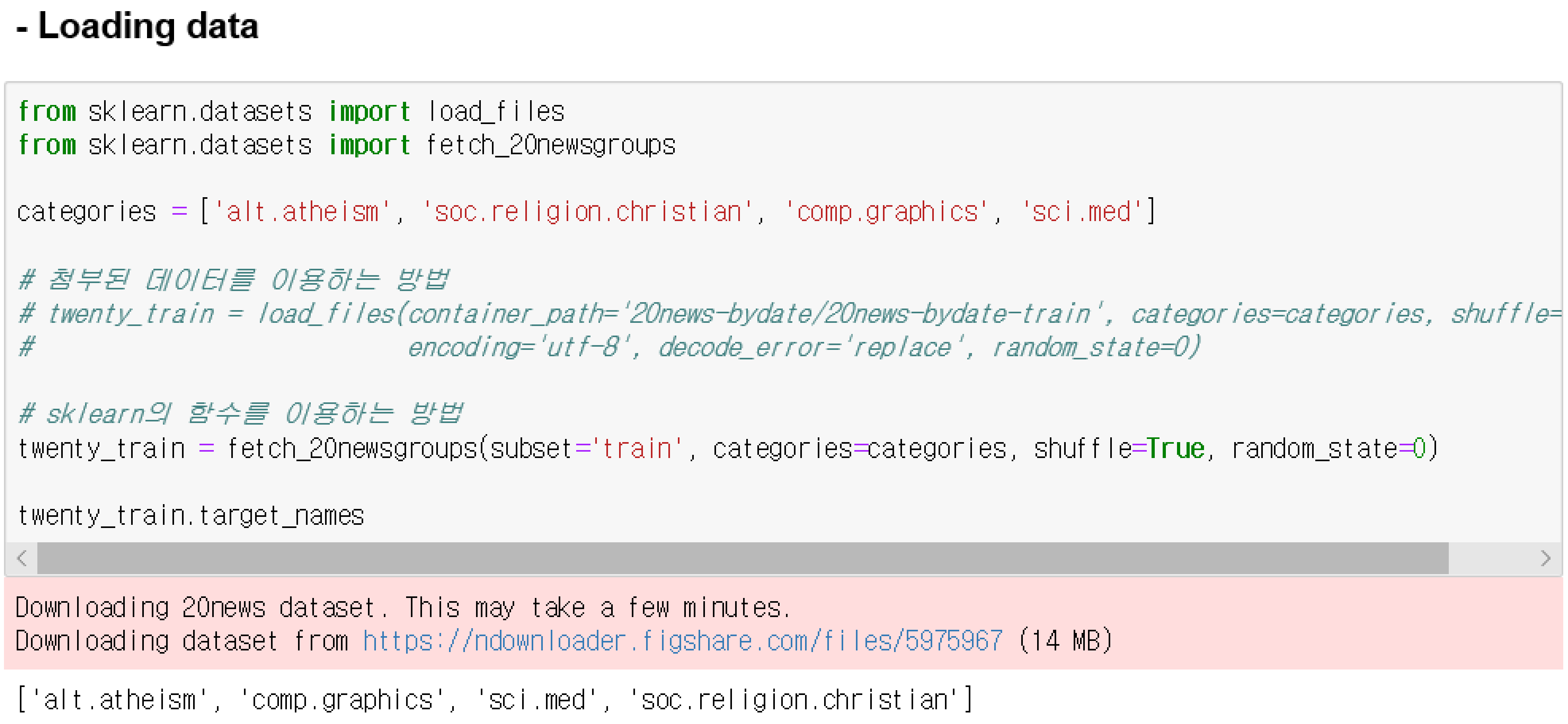
- 20 Newsgroups dataset
  - http://qwone.com/~jason/20Newsgroups/
  - 원본 데이터: 총 20개 카테고리로 분류된 약 20,000개 문서
  - 연관이 있는 분야와 아예 연관이 없는 분야로 카테고리 구성

  - 본 수업에서는 4개 카테고리의 일부 데이터만 사용 (카테고리마다 약 500개)
  - 첨부된 폴더 이용 혹은 sklearn의 함수 이용

  - https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

Loading data

- sklearn의 fetch_20newsgroups 함수 사용 [20 Newsgroups dataset 한정]
  - from sklearn.datasets import fetch_20newsgroups
  - train, test set 지정 및 분류 학습에 사용할 category 지정



- Loading data

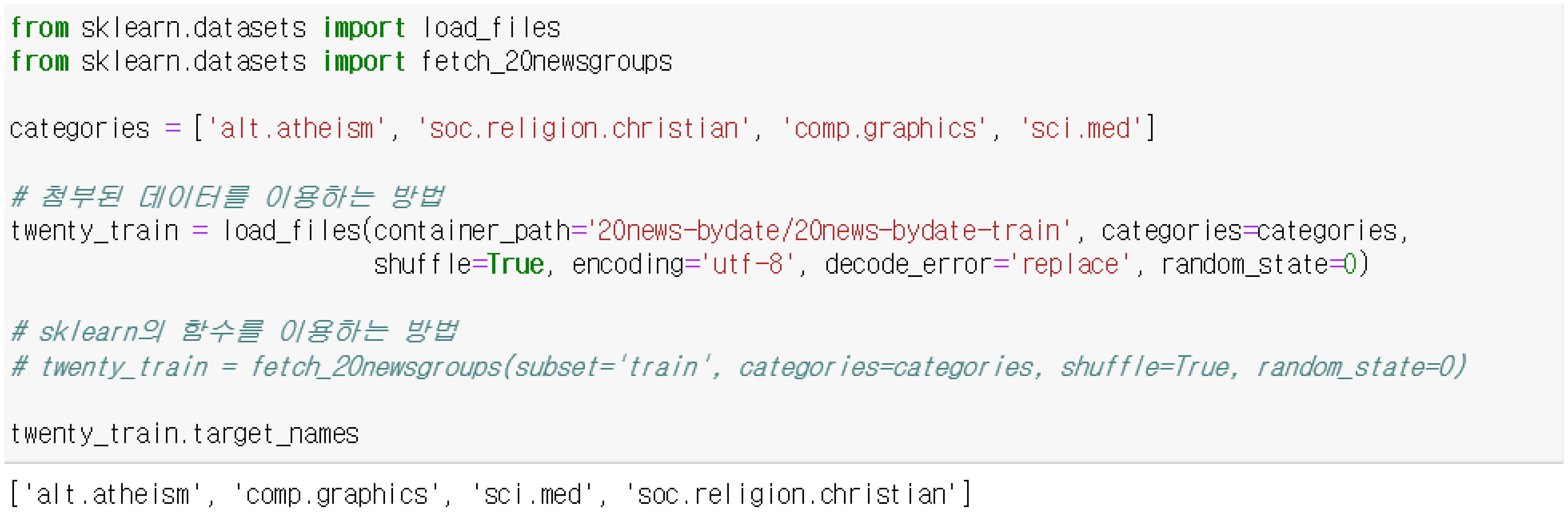
```
from sklearn.datasets import load_files
from sklearn.datasets import fetch_20newsgroups

categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']

# 첨부된 데이터를 이용하는 방법
# twenty_train = load_files(container_path='20news-bydate/20news-bydate-train', categories=categories, shuffle=
#                          encoding='utf-8', decode_error='replace', random_state=0)

# sklearn의 함수를 이용하는 방법
twenty_train = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_state=0)

twenty_train.target_names
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
```

```
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

Loading data

- sklearn의 load_files 함수 이용 [일반적인 dataset에 사용 가능]
  - from sklearn.datasets import load_files
  - container_path 하위 directory로 각 category명을 이름으로 갖는 폴더, 폴더 안에 각 파일들



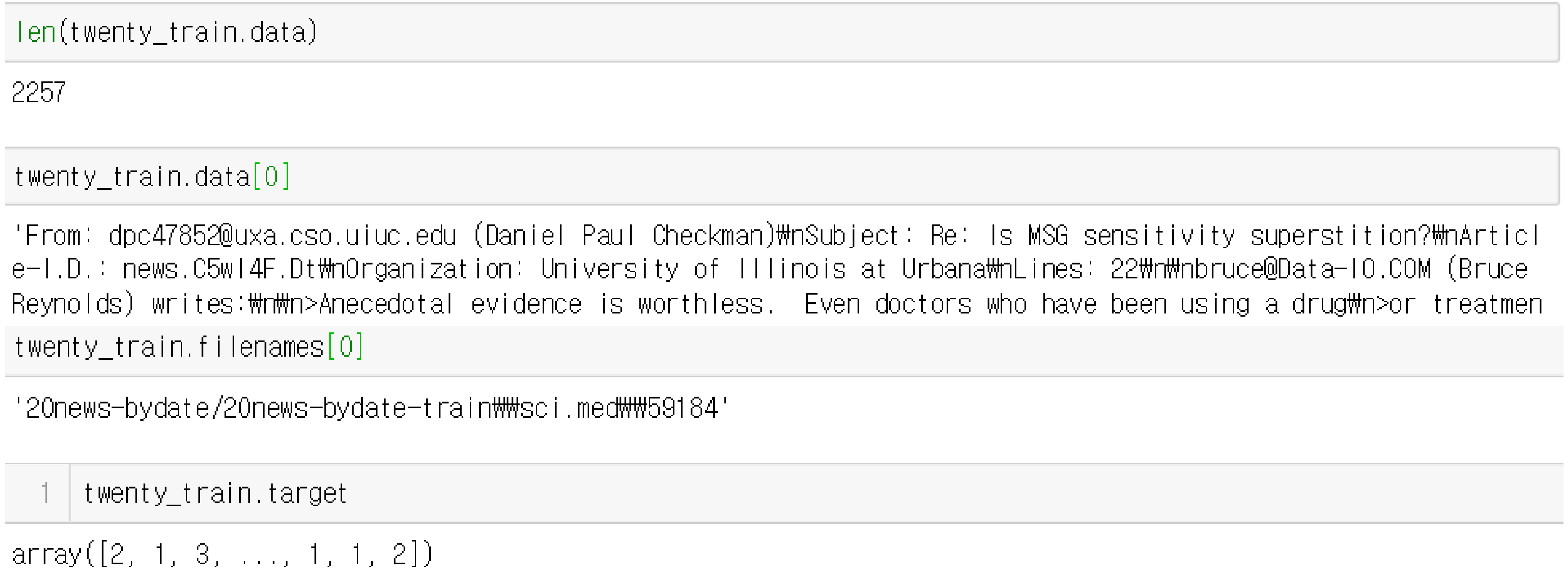
```python
from sklearn.datasets import load_files
from sklearn.datasets import fetch_20newsgroups

categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']

# 첨부된 데이터를 이용하는 방법
twenty_train = load_files(container_path='20news-bydate/20news-bydate-train', categories=categories,
                          shuffle=True, encoding='utf-8', decode_error='replace', random_state=0)

# sklearn의 함수를 이용하는 방법
# twenty_train = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_state=0)

twenty_train.target_names
```

```
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

```
container_folder/
  category_1_folder/
    file_1.txt file_2.txt ... file_42.txt

  category_2_folder/
    file_43.txt file_44.txt ...
```

Loading data

- Dataset attributes
  - data: Raw text data
  - target: Target labels, integer array
  - target_names: Names of target classes
  - DESCR: Full description of dataset (default = None)
  - filenames: Filenames holding the dataset



```
len(twenty_train.data)
```

```
2257
```

```
twenty_train.data[0]
```

```
'From: dpc47852@uxa.cso.uiuc.edu (Daniel Paul Checkman)₩nSubject: Re: Is MSG sensitivity superstition?₩nArticl
e-I.D.: news.C5wI4F.Dt₩nOrganization: University of Illinois at Urbana₩nLines: 22₩n₩nbruce@Data-IO.COM (Bruce
Reynolds) writes:₩n₩n>Anecedotal evidence is worthless.  Even doctors who have been using a drug₩n>or treatmen
```

```
twenty_train.filenames[0]
```

```
'20news-bydate/20news-bydate-train₩₩sci.med₩₩59184'
```

```
twenty_train.target
```

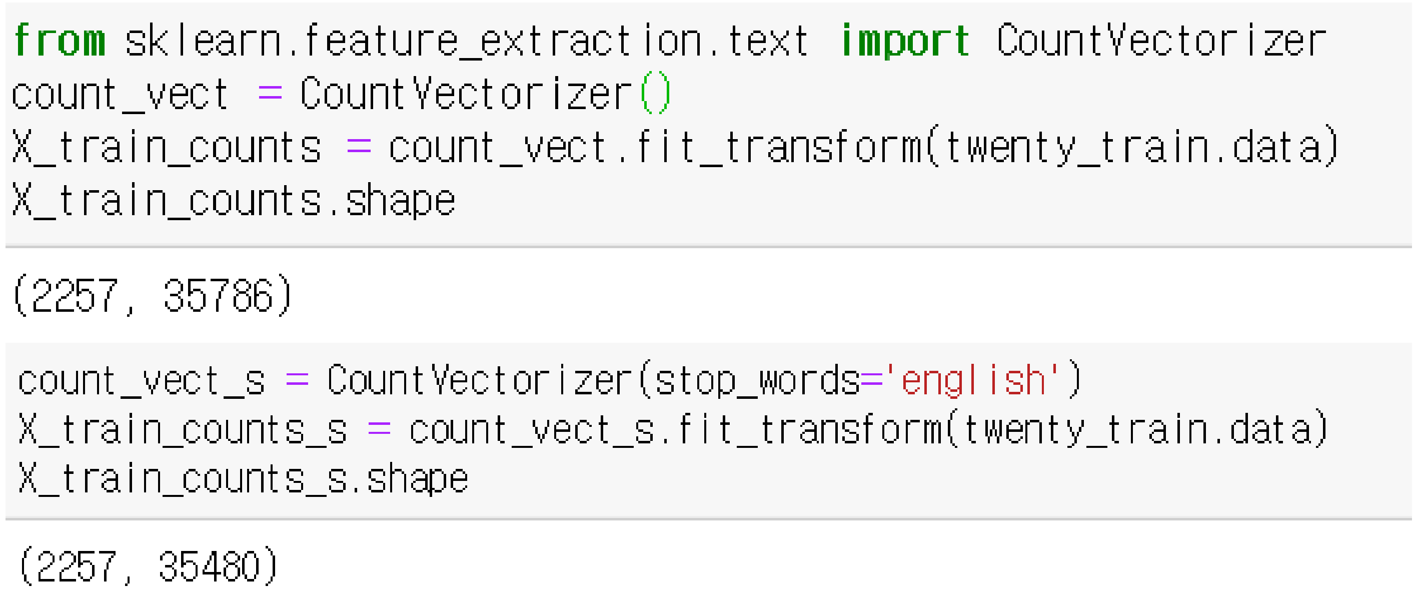
```
array([2, 1, 3, ..., 1, 1, 2])
```

Extracting Features

- Count Vectorizer
  - 단어의 출현 빈도(frequency)로 여러 문서들을 벡터화
- 작동방식
  - 토큰화(Tokenization)
    - 텍스트를 개별 단어(토큰)로 분리
    - Ex) “I love apples” => ["I", "love", "apples"]로 분리
  - 단어 카운트(Word Count)
    - 각 토큰의 빈도수 계산
    - Ex) "apple banana apple strawberry banana"라면, "apple"은 2, "banana"는 2, "strawberry"는 1로 계산
  - 벡터화(Vectorization)
    - 각 단어를 벡터 형태로 변환
  - 정규화 및 정제
    - 필요에 따라 Stop words 제거, 소문자 변환, 오타 수정 등 전처리 과정 수행가능

Extracting Features

- Count Vectorizer
  - from sklearn.feature_extraction.text import CountVectorizer
  - Tokenizing, stopwords filtering, word count, n-gram 까지 모두 처리 가능
  - 기본 default: 모두 Lowercase로 convert, stopwords 미처리, n-gram 미사용, 모든 단어 사용
  - stop_words: ‘english’ 사용 시 built-in stop word list를 제외하고 구성
  - ngram_range: (min_n, max_n) / max_df / min_df / max_features



```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)
X_train_counts.shape
```

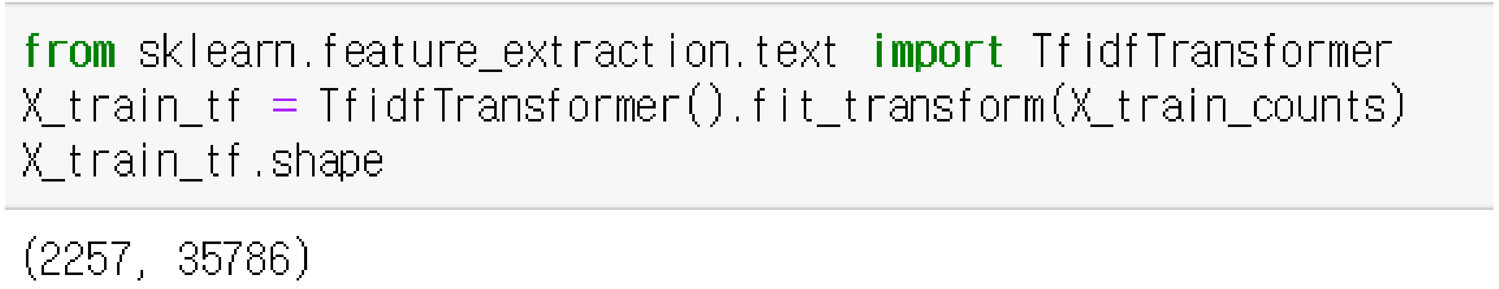
```
(2257, 35786)
```

```
count_vect_s = CountVectorizer(stop_words='english')
X_train_counts_s = count_vect_s.fit_transform(twenty_train.data)
X_train_counts_s.shape
```

```
(2257, 35480)
```

Extracting Features

- Tf-idf(Term Frequency, Inverse Document Frequency) Transformer
  - Term Frequency = $TF(t,d) \times IDF(t,D)$
    - 특정 단어의 등장 빈도
    - $TF(t,d) = \frac{n_{t,d}}{\sum_k n_{k,d}} = \frac{\text{문서 } d \text{ 내에서 단어 } t \text{의 등장 횟수}}{\text{문서 } d \text{ 내의 모든 단어의 총 등장 횟수의 합계}}$
  - Inverse Document Frequency
    - $IDF(t,D) = \log\left(\frac{N}{1+|\{d \in D L t \in d\}|}\right) = log \frac{\text{전체 문서의 수}}{\text{단어 } t \text{가 포함된 문서의 수}}$
  - from sklearn.feature_extraction.text import TfidfTransformer
  - Count Vector을 받아 Tf-idf 반환



```
from sklearn.feature_extraction.text import TfidfTransformer
X_train_tf = TfidfTransformer().fit_transform(X_train_counts)
X_train_tf.shape
```

```
(2257, 35786)
```

Training Classifier

- Naïve Bayes classifier
  - from sklearn.naive_bayes import (사용할 NB 모델)
  - .fit()으로 training data에 학습시킨 후 새로운 input에 대해선 .predict()로 분류

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB().fit(X_train_tfidf, twenty_train.target)

docs_new = ['cancer patient', 'OpenGL on the GPU is fast']
X_new_counts = count_vect.transform(docs_new)
X_new_tfidf = tfidf_transformer.transform(X_new_counts)

predicted = nb.predict(X_new_tfidf)

for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, twenty_train.target_names[category]))
```

```
'cancer patient' => sci.med
'OpenGL on the GPU is fast' => comp.graphics
```

Training Classifier

- Naïve Bayes classifier
  - GaussianNB, MultinomialNB, BernoulliNB
  - GaussianNB

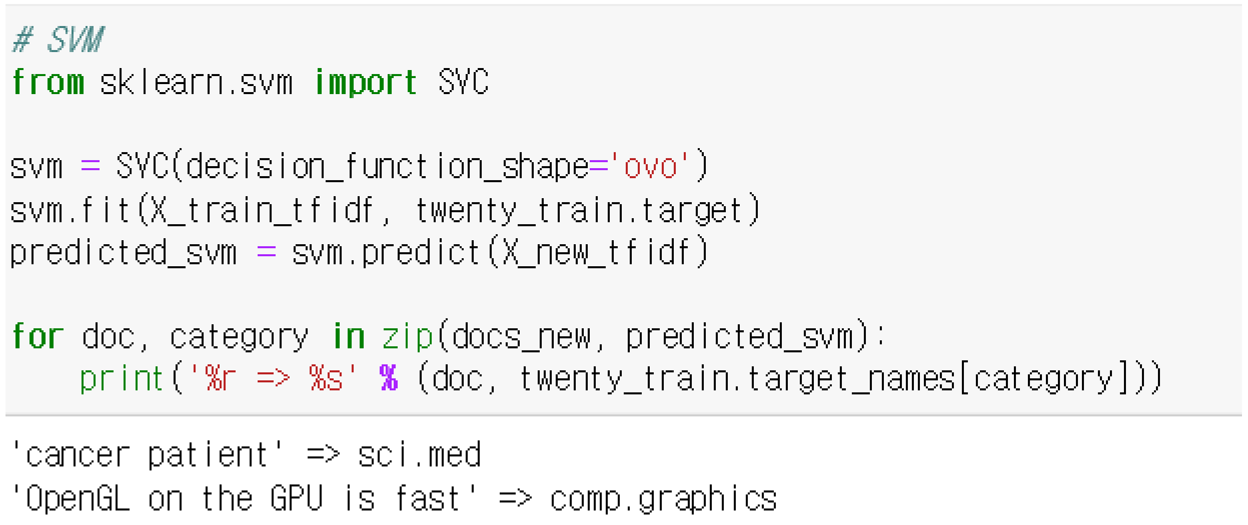
  http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
  - MultinomialNB

  http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
  - BernoulliNB

  http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html

Training Classifier

- SVM classifier
  - from sklearn.svm import (사용할 SVM class)
  - Naive bayes classifier와 비슷한 방식으로 사용하면 됨

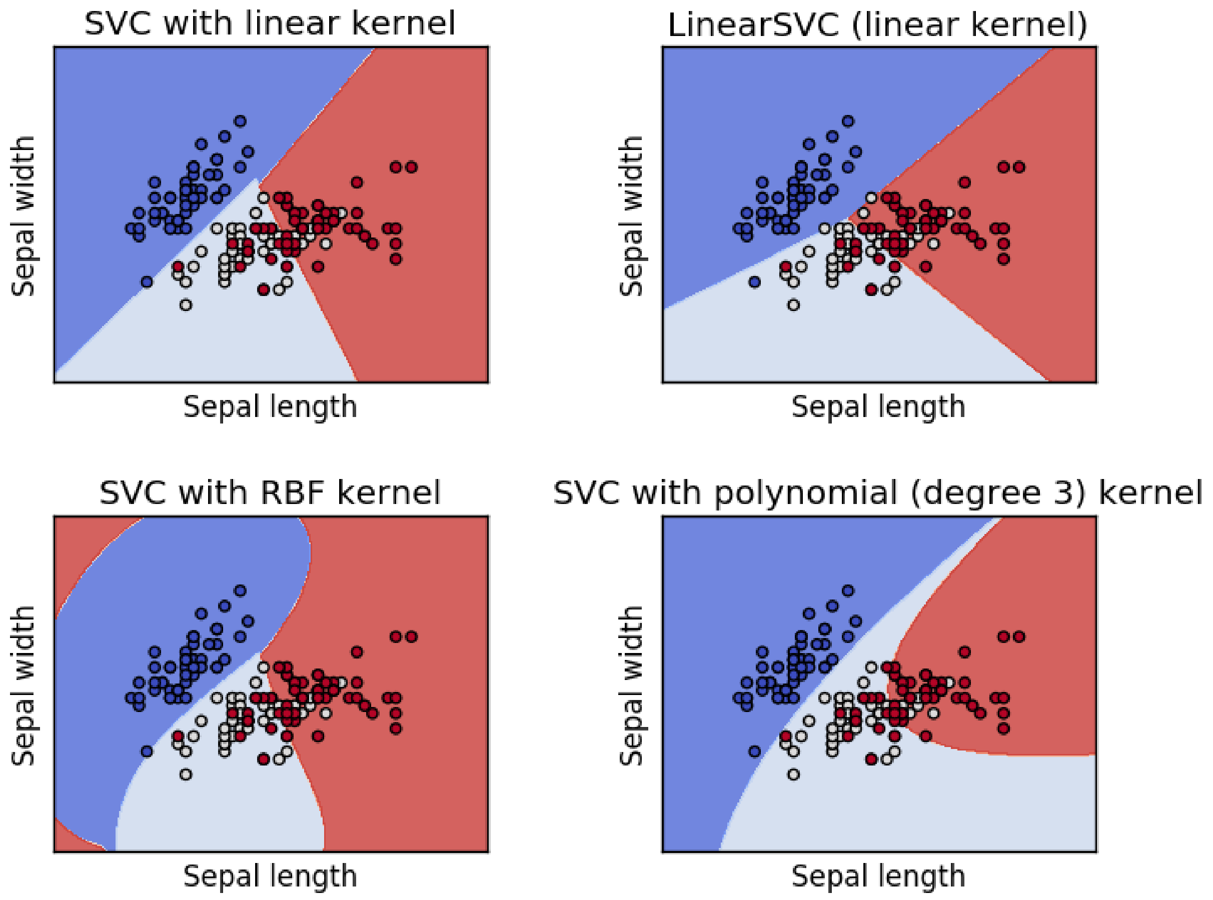


```
# SVM
from sklearn.svm import SVC

svm = SVC(decision_function_shape='ovo')
svm.fit(X_train_tfidf, twenty_train.target)
predicted_svm = svm.predict(X_new_tfidf)

for doc, category in zip(docs_new, predicted_svm):
    print('%r => %s' % (doc, twenty_train.target_names[category]))
```

```
'cancer patient' => sci.med
'OpenGL on the GPU is fast' => comp.graphics
```

Training Classifier

- SVM classifier
  - http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
  - SVC, NuSVC, LinearSVC 총 3개의 클래스 중 하나를 선택해서 사용 가능
    - SVC와 NuSVC는 동일하나 NuSVC에서는 파라미터 ν가 추가되어 training error의 upper bound와 support vector 비율의 lower bound를 제약식에 추가
    - LinearSVC: linear kernel만 사용



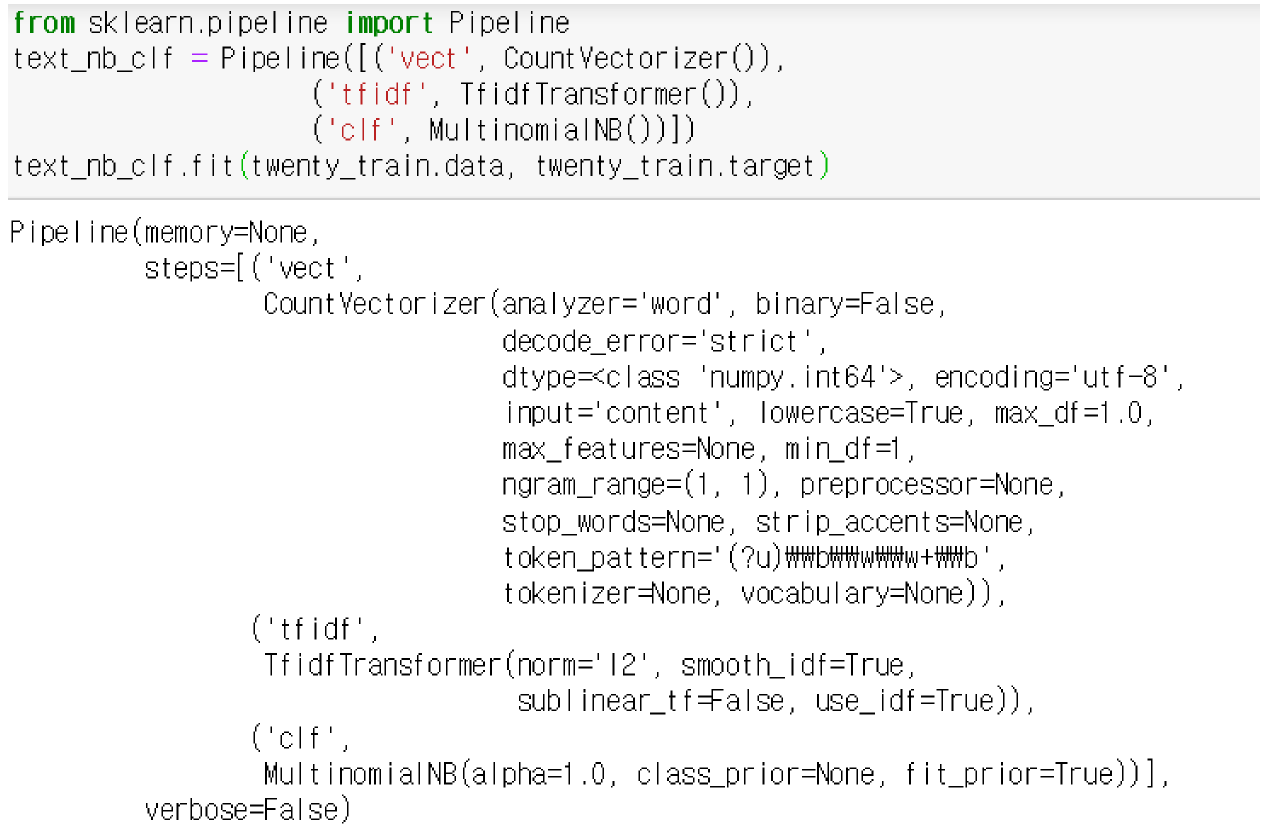
Training Classifier

- Multi-class classification
  - SVC, NuSVC: one-vs-one (OVO) 방법론 사용
    - n_class * (n_class – 1) / 2
    - decision_function_shape 옵션을 변경하여 각각의 classifier 결과를 어떻게 aggregate할 것인지 변경 가능
  - LinearSVC: one-vs-rest (OVR) 방법론 사용
    - n_class

- SVM parameters
  - kernel(string): linear, poly, rbf, sigmoid, precomputed
  - probability(Boolean): True/False
  - C: Penalty parameter, default=1.0
  - degree: Degree of the polynomial kernel function
  - gamma: Kernel coefficient(rbf, poly, sigmoid)
  - coef0: Decision function의 weight
  - Intercept_: Decision function의 계수 ....

Building a pipeline

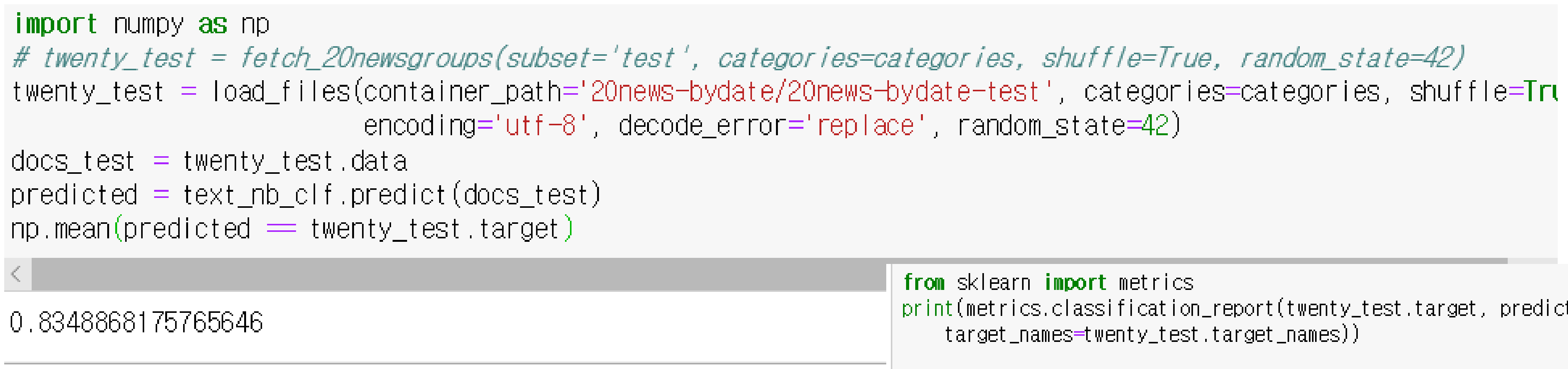
- Pipeline
  - from sklearn.pipeline import Pipeline
  - Text processing, transforming, classification까지의 과정을 하나로 묶어 처리할 수 있는 pipeline 기능 제공



```python
from sklearn.pipeline import Pipeline
text_nb_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', MultinomialNB())])
text_nb_clf.fit(twenty_train.data, twenty_train.target)
```

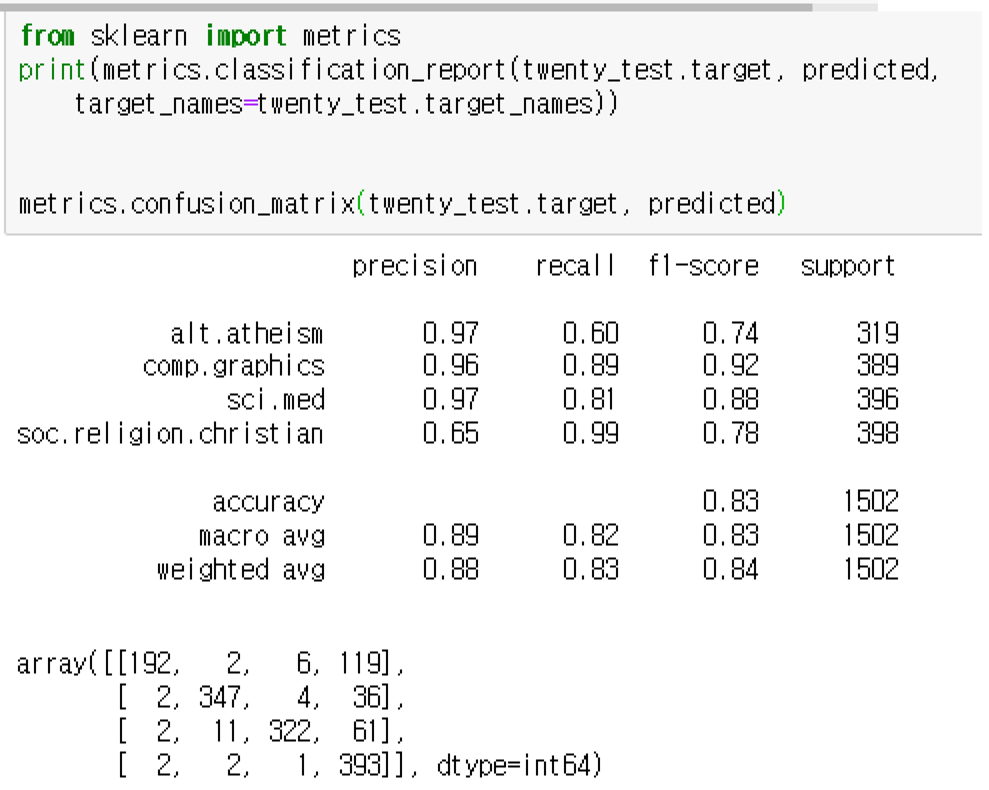
```
Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)₩₩b₩₩w₩₩w+₩₩b',
                                 tokenizer=None, vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf=True)),
                ('clf',
                 MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))],
         verbose=False)
```

Evaluation



```
import numpy as np
# twenty_test = fetch_20newsgroups(subset='test', categories=categories, shuffle=True, random_state=42)
twenty_test = load_files(container_path='20news-bydate/20news-bydate-test', categories=categories, shuffle=Tru
                         encoding='utf-8', decode_error='replace', random_state=42)
docs_test = twenty_test.data
predicted = text_nb_clf.predict(docs_test)
np.mean(predicted == twenty_test.target)
```

```
0.8348868175765646
```



```
from sklearn import metrics
print(metrics.classification_report(twenty_test.target, predicted,
    target_names=twenty_test.target_names))


metrics.confusion_matrix(twenty_test.target, predicted)
```

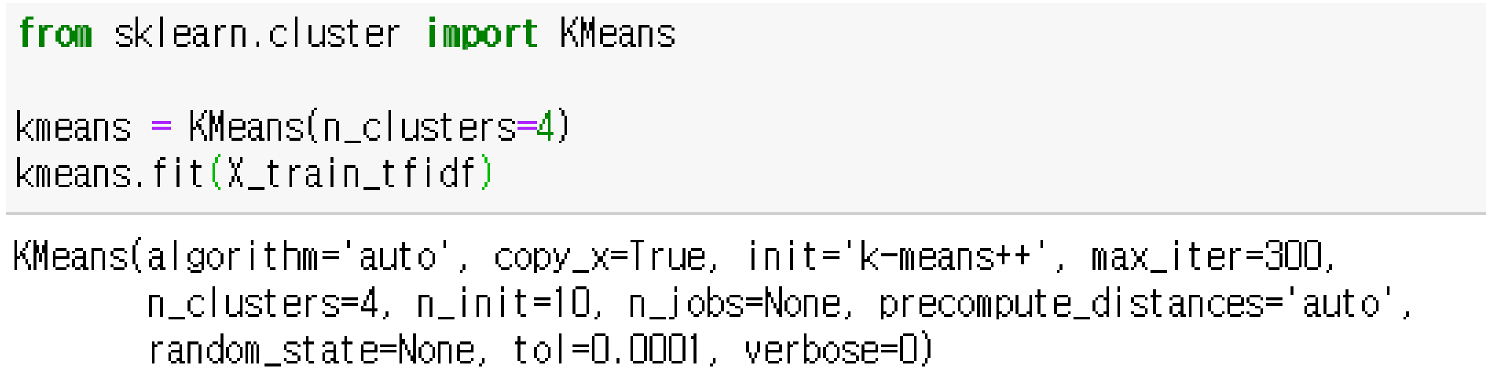
```
                        precision    recall  f1-score   support

           alt.atheism       0.97      0.60      0.74       319
         comp.graphics       0.96      0.89      0.92       389
               sci.med       0.97      0.81      0.88       396
soc.religion.christian       0.65      0.99      0.78       398

              accuracy                           0.83      1502
             macro avg       0.89      0.82      0.83      1502
          weighted avg       0.88      0.83      0.84      1502


array([[192,   2,   6, 119],
       [  2, 347,   4,  36],
       [  2,  11, 322,  61],
       [  2,   2,   1, 393]], dtype=int64)
```

Document Classification and Clustering Using Python

PART 2. CLUSTERING

Clustering

- K-means Clustering
  - from sklearn.cluster import KMeans
  - parameter: n_clusters



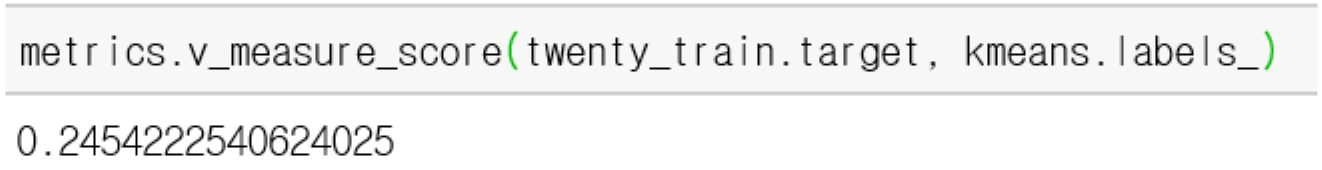
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4)
kmeans.fit(X_train_tfidf)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

- Agglomerative Clustering
  - from sklearn.cluster import AgglomerativeClustering()
  - parameter: n_clusters
  - linkage: ward, complete, average

Clustering

- Evaluation metrics
  - from sklearn.metrics import [사용 metric 이름]
  - Homogeneity_score: 각 군집이 하나의 class만을 갖는지
  - Completeness_score: 한 class의 모든 멤버가 같은 군집에 속하는지
  - V_measure_score: homogeneity_score와 completeness_score의 조화 평균
  - Adjusted_rand_score
  - Adjusted_mutual_info_score
  - Fowlkes_mallows_score
  - Silhouette_score

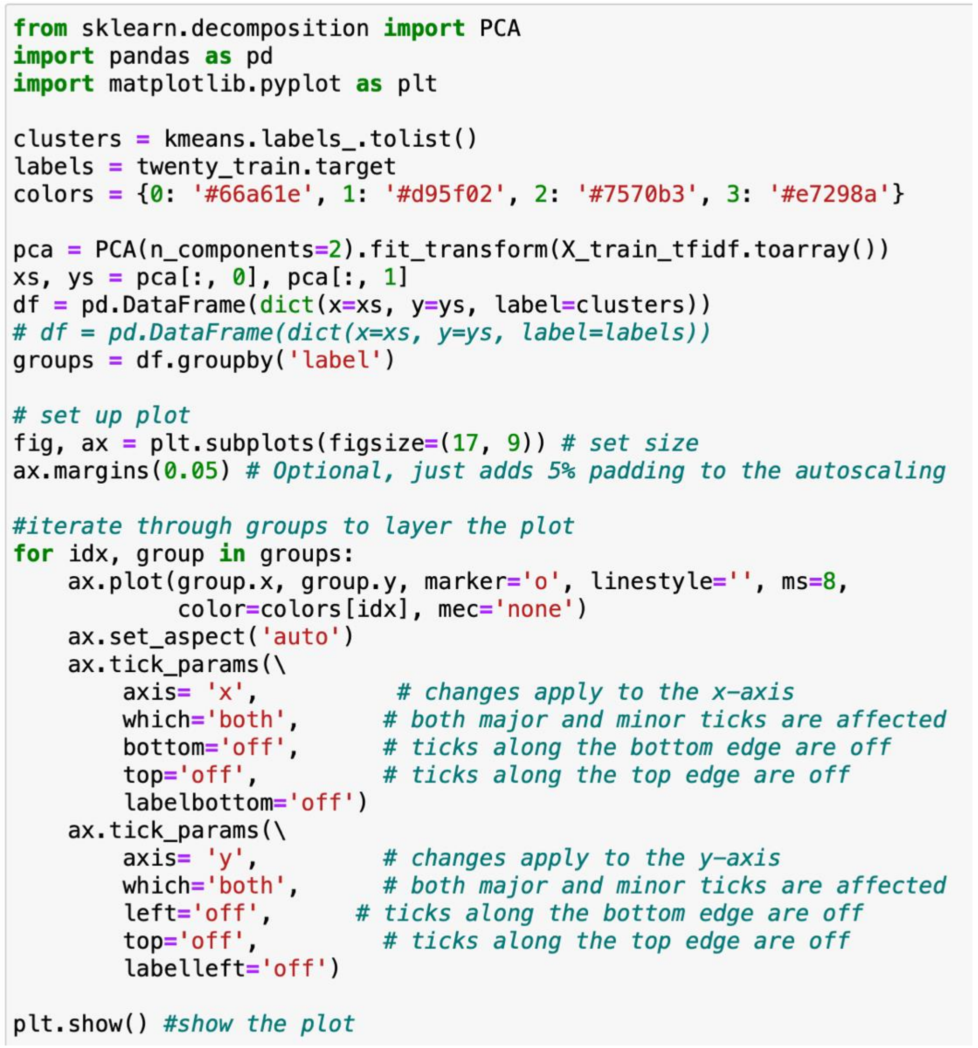


```
metrics.v_measure_score(twenty_train.target, kmeans.labels_)
```

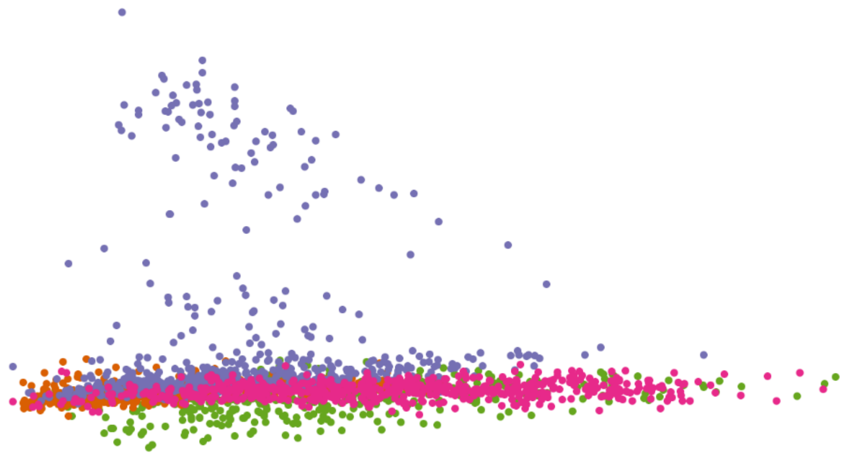
```
0.2454222540624025
```

Clustering

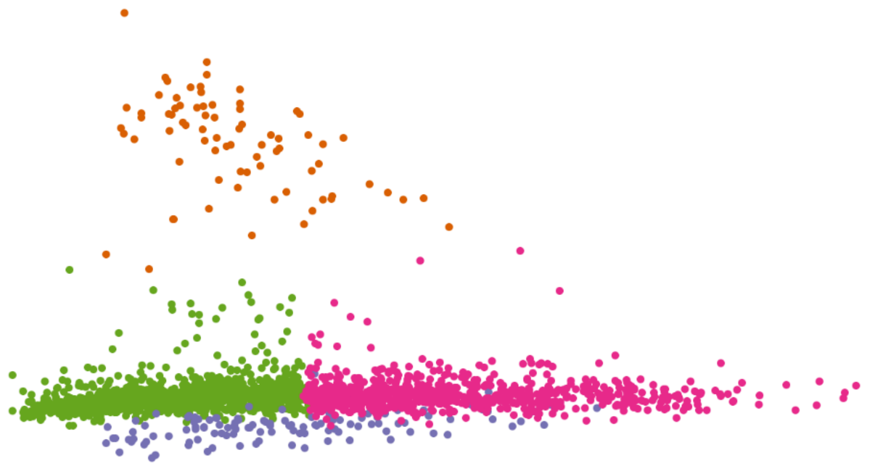
- 차원 축소를 통한 시각화(PCA)



```python
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt

clusters = kmeans.labels_.tolist()
labels = twenty_train.target
colors = {0: '#66a61e', 1: '#d95f02', 2: '#7570b3', 3: '#e7298a'}

pca = PCA(n_components=2).fit_transform(X_train_tfidf.toarray())
xs, ys = pca[:, 0], pca[:, 1]
df = pd.DataFrame(dict(x=xs, y=ys, label=clusters))
# df = pd.DataFrame(dict(x=xs, y=ys, label=labels))
groups = df.groupby('label')

# set up plot
fig, ax = plt.subplots(figsize=(17, 9)) # set size
ax.margins(0.05) # Optional, just adds 5% padding to the autoscaling

#iterate through groups to layer the plot
for idx, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=8,
            color=colors[idx], mec='none')
    ax.set_aspect('auto')
    ax.tick_params(\
        axis= 'x',          # changes apply to the x-axis
        which='both',      # both major and minor ticks are affected
        bottom='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelbottom='off')
    ax.tick_params(\
        axis= 'y',         # changes apply to the y-axis
        which='both',      # both major and minor ticks are affected
        left='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelleft='off')

plt.show() #show the plot
```



실제 정답 labeling



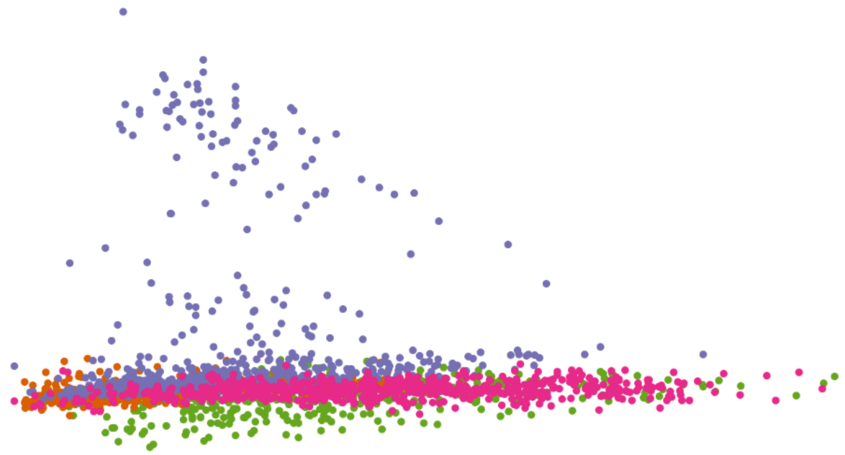
Clustering 결과

Clustering

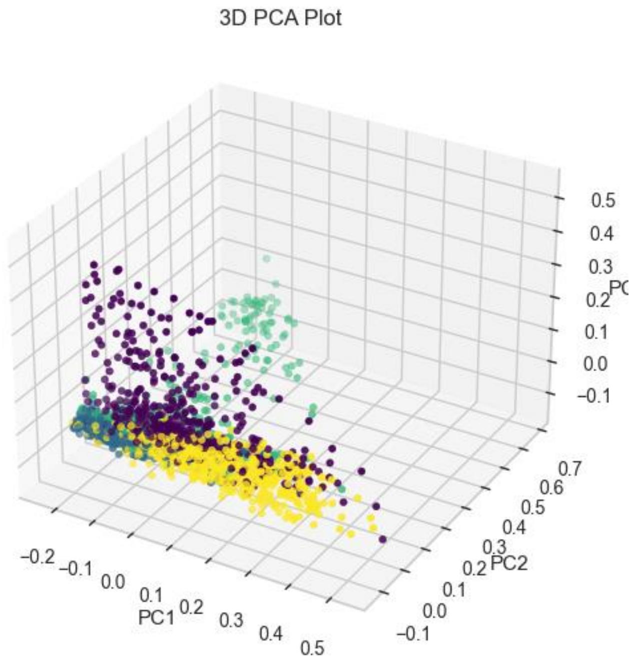
- 차원 축소를 통한 시각화(3D Plot)

```
from mpl_toolkits.mplot3d import Axes3D


X_pca = PCA(n_components=3).fit_transform(X_train_tfidf.toarray())
# df = pd.DataFrame(dict(x=xs, y=ys, label=labels))
labels = twenty_train.target
```

```
# Creating a 3D scatter plot
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111, projection='3d')

# Plotting the data points
ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=labels, s=15, cmap='viridis', marker='o')

# Setting labels
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')

# Title
ax.set_title('3D PCA Plot')

# Showing the plot
plt.show()
```



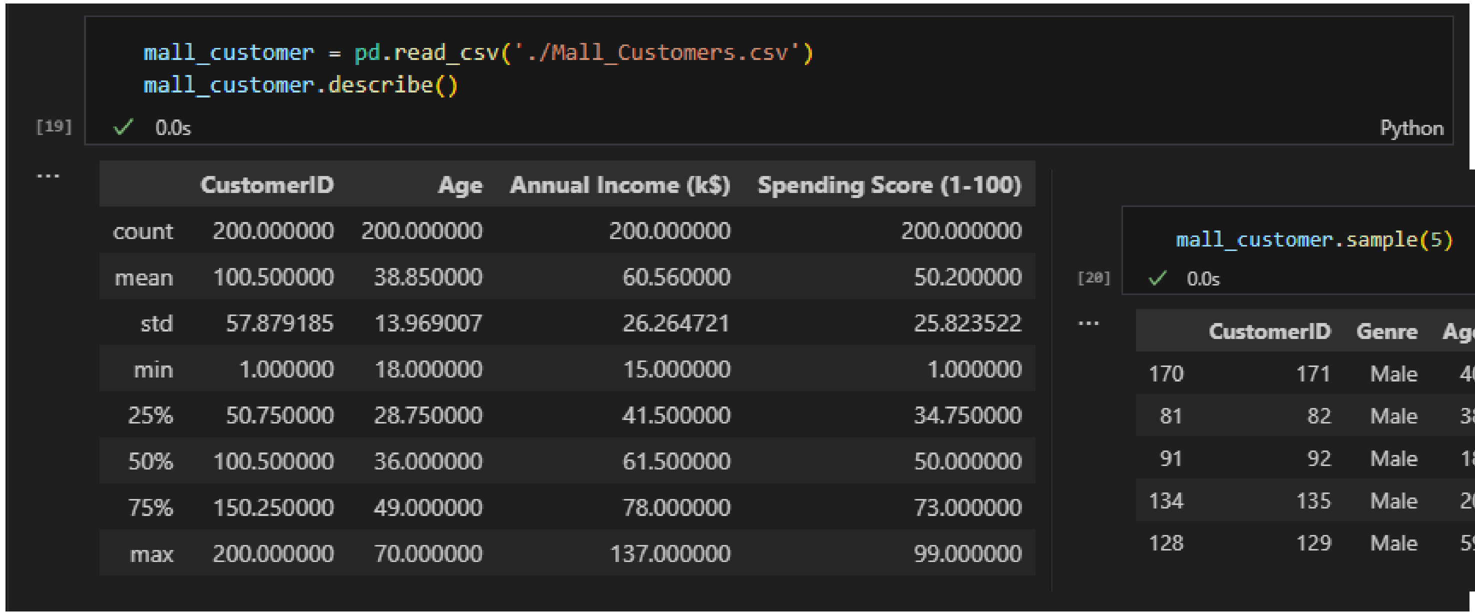
실제 정답 labeling



3D Plot 결과

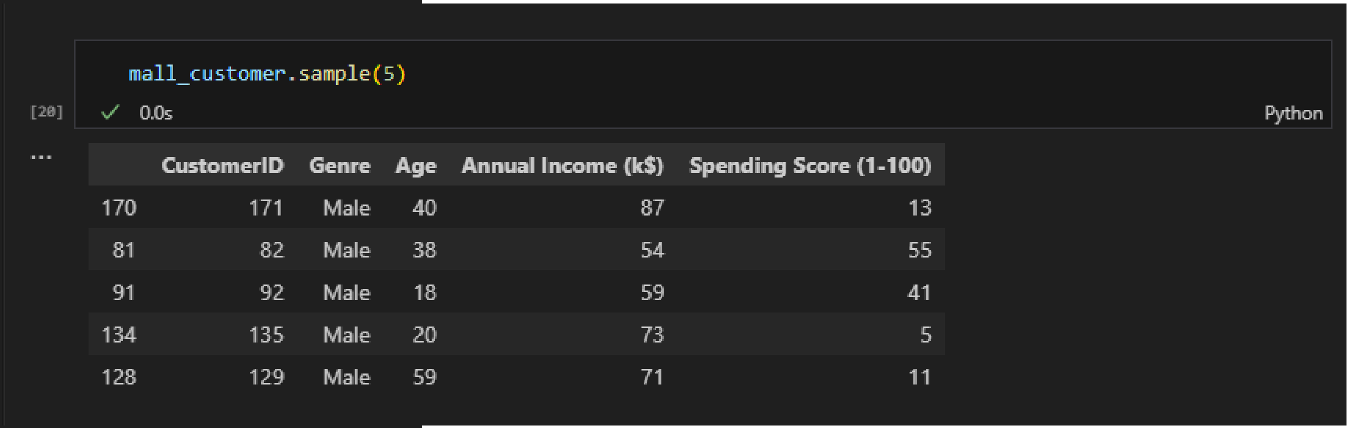
Clustering – Elbow Method

- 적절한 Cluster 개수 찾기
  - 실습 Dataset : mall customer (https://www.kaggle.com/datasets/shwetabh123/mall-customers)
    - 소득 및 소비 패턴에 따른 고객 군집화
    - 총 4개의 Features(Genre, Age, Annual income, Spending Score)
      - 분석목적에 맞는 Feature 선정 : [Annual Income, Spending_Score]



```
[19]
mall_customer = pd.read_csv('./Mall_Customers.csv')
mall_customer.describe()
✓ 0.0s                                                   Python
```

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000



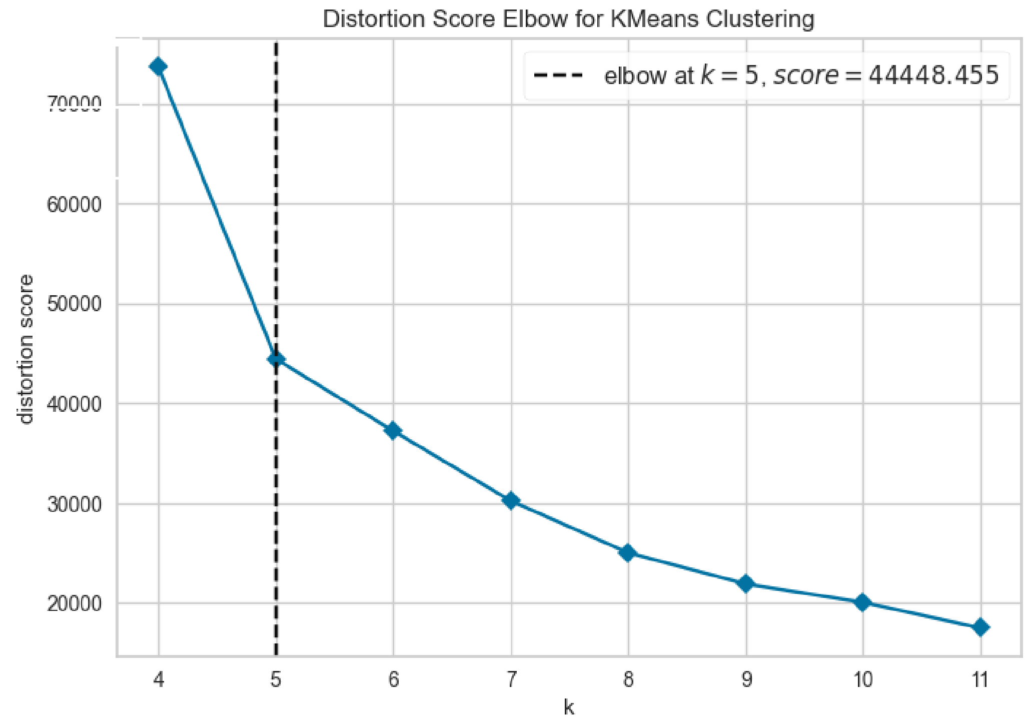
```
[20]
mall_customer.sample(5)
✓ 0.0s                                                   Python
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
170	171	Male	40	87	13
81	82	Male	38	54	55
91	92	Male	18	59	41
134	135	Male	20	73	5
128	129	Male	59	71	11

Clustering – Elbow Method

- Elbow Method
  - Idea: Score 변화량이 가장 큰 cluster가 Optimal
  - Distortion Score $D(k) = \sum_{i=1}^{n} \min_{\mu_j \in C}(\|x_i - \mu_j\|^2)$
    - n : 데이터 개수
    - C : 클러스터 집합
    - x_i: i번째 데이터 포인트
    - μ_j: j번째 클러스터의 중심(Centroid)
    - $\|x_i - \mu_j\|$: 데이터 포인트 x_i와 j번째 클러스터간 유클리드 거리

```
from yellowbrick.cluster import KElbowVisualizer
import matplotlib.pyplot as plt

X = mall_customer.iloc[:, [3, 4]].values

model = KMeans(n_init=10, random_state=42)
visualizer = KElbowVisualizer(
    model, k=(4,12), timings=False
)

visualizer.fit(X)# Fit the data to the visualizer
visualizer.show()
```



PROJECT#2

End of the Document