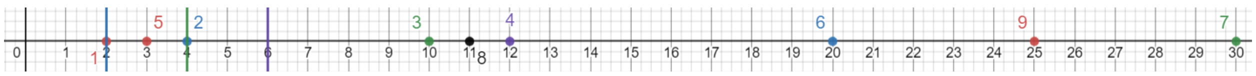
Data Mining (CSE542)

Homework 04

ID: __ Name: __조원석_ Date: __2023-05-08__

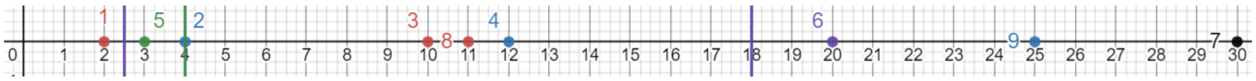
Task-1

Given the following points: 2, 4, 10, 12, 3, 20, 30, 11, 25. Assume $k = 3$, and that we randomly pick the initial means $\mu_1 = 2$, $\mu_2 = 4$ and $\mu_3 = 6$. Show the clusters obtained using K-means algorithm after one iteration, and show the new means for the next iteration.



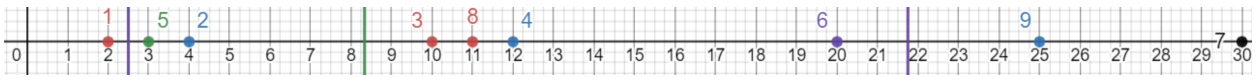
Initial means => $\mu_1 = 2, \mu_2 = 4, \mu_3 = 6$

Until assume k=3, I do that.



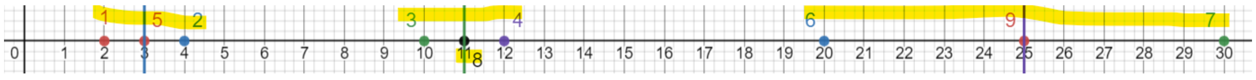
Closest mean_1 : C1 = {2, 3} C2 = {4}, C3{10, 11, 12, 20, 25, 30}

New means_1 => $\mu_1 = \frac{2+3}{2} = 2.5, \mu_2 = 4, \mu_3 = \frac{10+11+12+20+25+30}{6} = 18$



2nd iteration : C1 = {2, 3} C2 = {4, 10, 11}, C3{ 12, 20, 25, 30 }

New means_2 => $\mu_1 = \frac{2+3}{2} = 2.5, \mu_2 = \frac{4+10+11}{3} = 8.33, \mu_3 = \frac{12+20+25+30}{4} = 21.75$



3rd iteration : C1 = {2, 3, 4} C2 = {10, 11, 12}, C3{ 20, 25, 30 }

New means_3 => $\mu_1 = \frac{2+3+4}{3} = 3, \mu_2 = \frac{10+11+12}{3} = 11, \mu_3 = \frac{20+25+30}{3} = 25$

So answer is $\mu_1 = 3, \mu_2 = 11, \mu_3 = 25$

Task-2

Given the two-dimensional points in Table 13.2, assume that $k = 2$, and that initially the points are assigned to clusters as follows: $C_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4\}$ and $C_2 = \{\mathbf{x}_3, \mathbf{x}_5\}$.

Table 13.2. Dataset

	X_1	X_2
$\mathbf{x}_1^T$	0	2
$\mathbf{x}_2^T$	0	0
$\mathbf{x}_3^T$	1.5	0
$\mathbf{x}_4^T$	5	0
$\mathbf{x}_5^T$	5	2

Apply the K-means algorithm until convergence, that is, the clusters do not change, assuming (1) the usual Euclidean distance or the L_2-*norm* as the distance between points, defined as $\|\mathbf{x}_i - \mathbf{x}_j\|_2 = \left(\sum_{a=1}^{d} (x_{ia} - x_{ja})^2\right)^{1/2}$, and (2) the Manhattan distance or the L_1-*norm* defined as $\|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{a=1}^{d} |x_{ia} - x_{ja}|$.

- **Eucildean distance**

	$d(x_i, \mu_1)$	$d(x_i, \mu_2)$	Cluster
x_1	2.1	3.4	C_1
x_2	1.8	3.4	C_1
x_3	0.7	2.0	C_1
x_4	3.4	2.0	C_2
x_5	3.6	2.0	C_2

	$d(x_i, \mu_1)$	$d(x_i, \mu_2)$	Cluster
x_1	1.42	5.1	C_1
x_2	0.83	5.1	C_1
x_3	1.2	3.6	C_1
x_4	4.5	1.0	C_2
x_5	4.7	1.0	C_2

- **Manhattan distance**

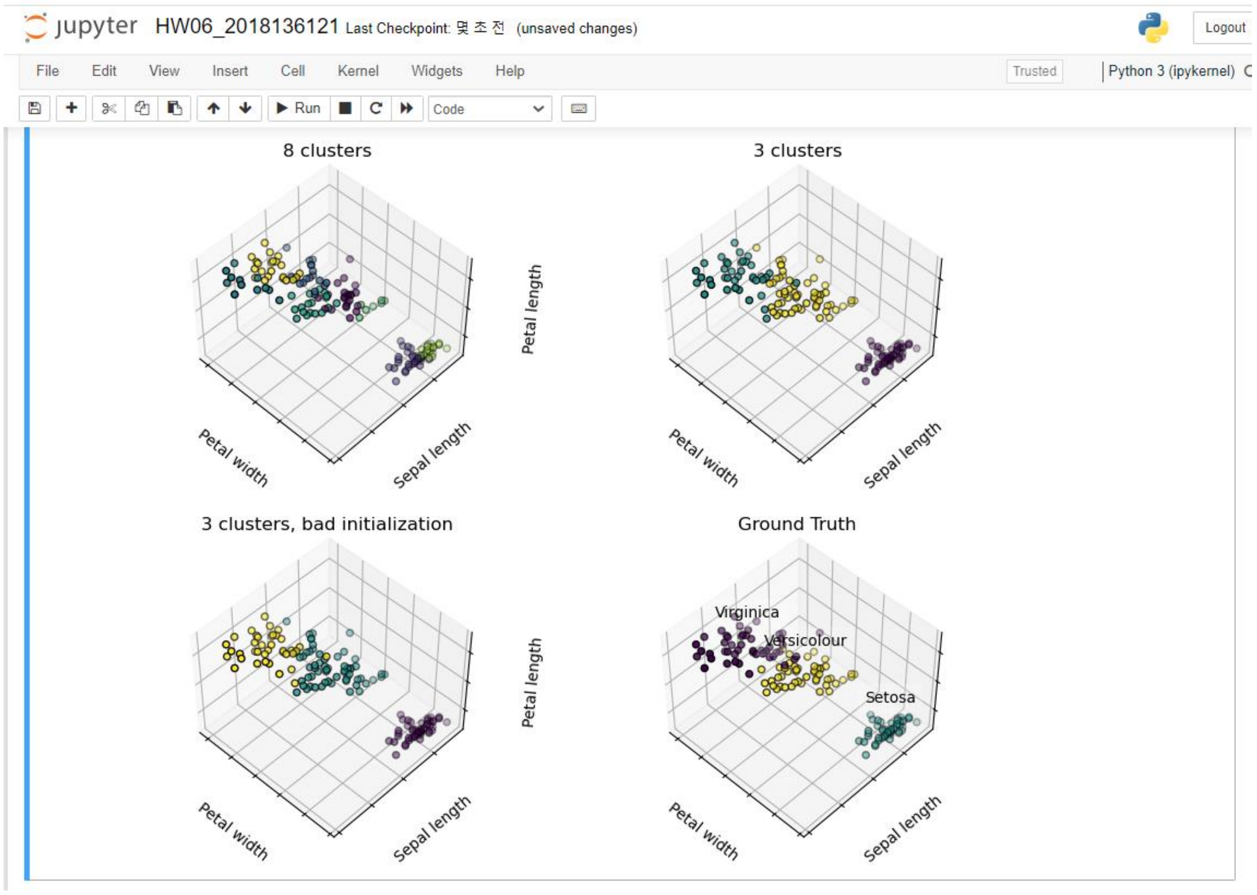
	$d(x_i, \mu_1)$	$d(x_i, \mu_2)$	**Cluster**
x_1	**3**	**4.25**	C_1
x_2	**2.34**	**4.25**	C_1
x_3	**0.84**	**2.75**	C_1
x_4	**4**	**2.75**	C_2
x_5	**4.66**	**2.75**	C_2

	$d(x_i, \mu_1)$	$d(x_i, \mu_2)$	**Cluster**
x_1	**1.83**	**6**	C_1
x_2	**1.17**	**6**	C_1
x_3	**1.67**	**4.5**	C_1
x_4	**5.17**	**1.0**	C_2
x_5	**5.83**	**1.0**	C_2

Task-3. Run the code and record the results for the following examples from the site

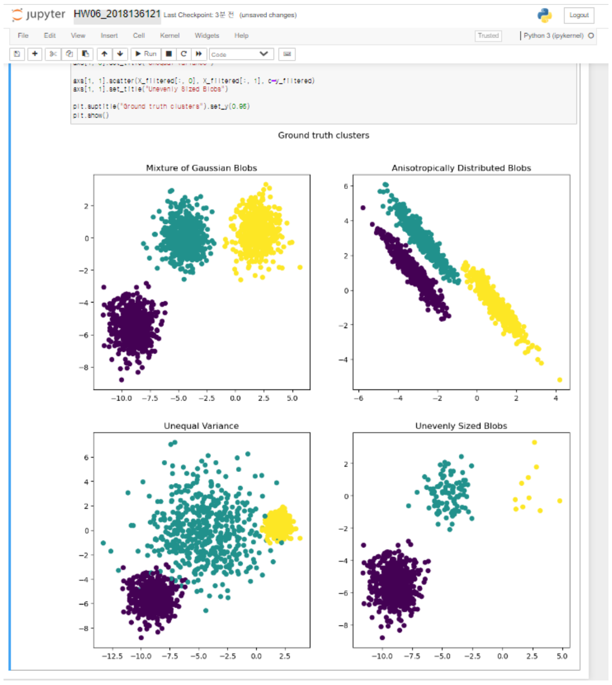
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

1. K-means Clustering



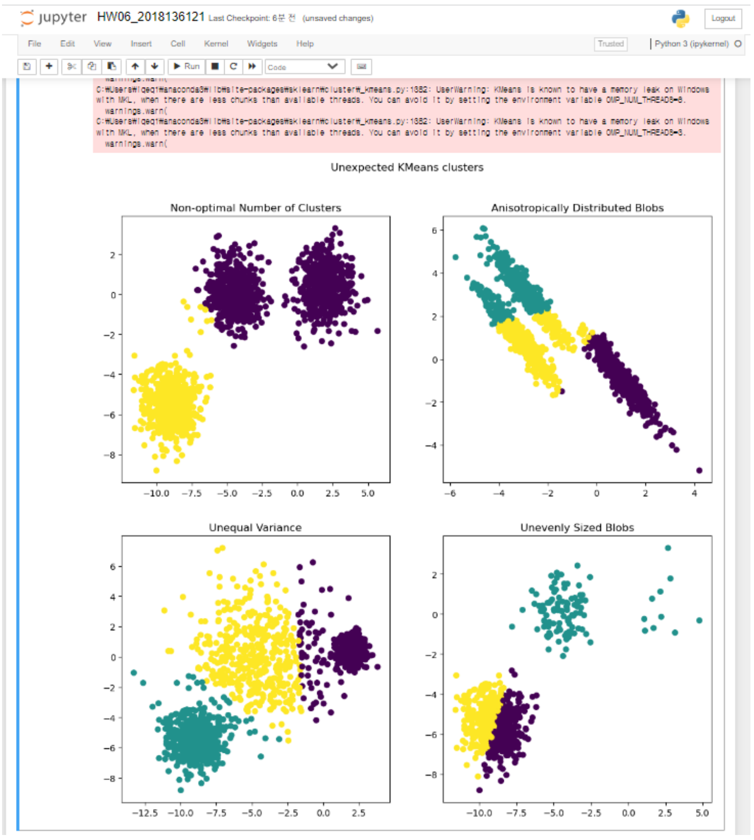
2. Demonstration of k-means assumptions

- Data generation



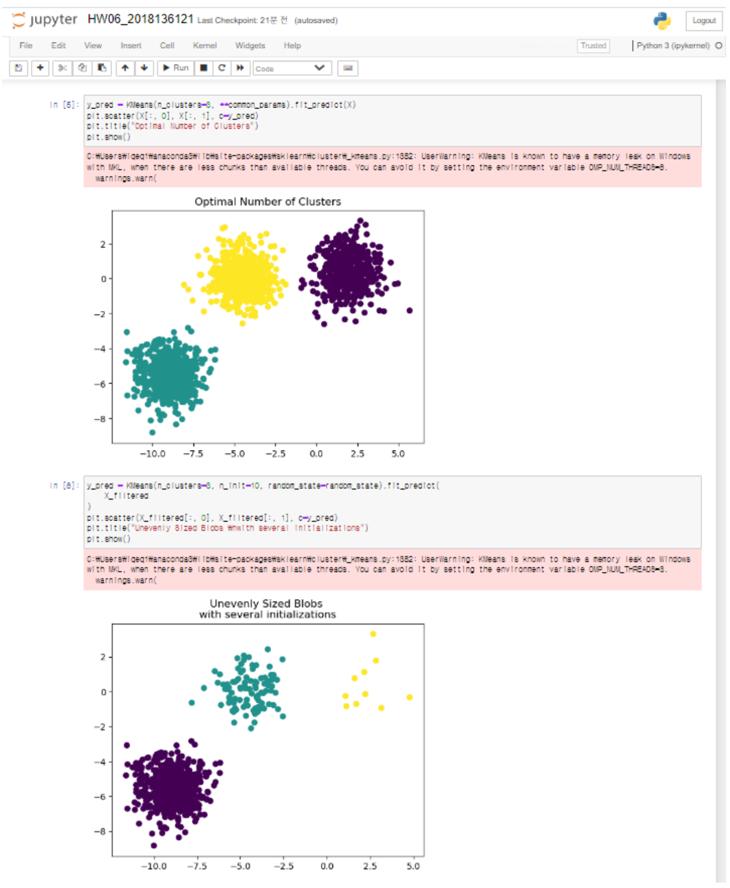
```
axs[1, 1].scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_filtered)
axs[1, 1].set_title("Unevenly Sized Blobs")

plt.suptitle("Ground truth clusters").set_y(0.95)
plt.show()
```

- Fit model and flot results



```
C:#Users#lqeq1#anaconda3#lib#site-packages#sklearn#cluster#_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=6.
  warnings.warn(
C:#Users#lqeq1#anaconda3#lib#site-packages#sklearn#cluster#_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```

jupyter
HW06_2018136121 Last Checkpoint: 6분 전 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (ipykernel)
Run
Code
Unexpected KMeans clusters
Non-optimal Number of Clusters
Anisotropically Distributed Blobs
Unequal Variance
Unevenly Sized Blobs

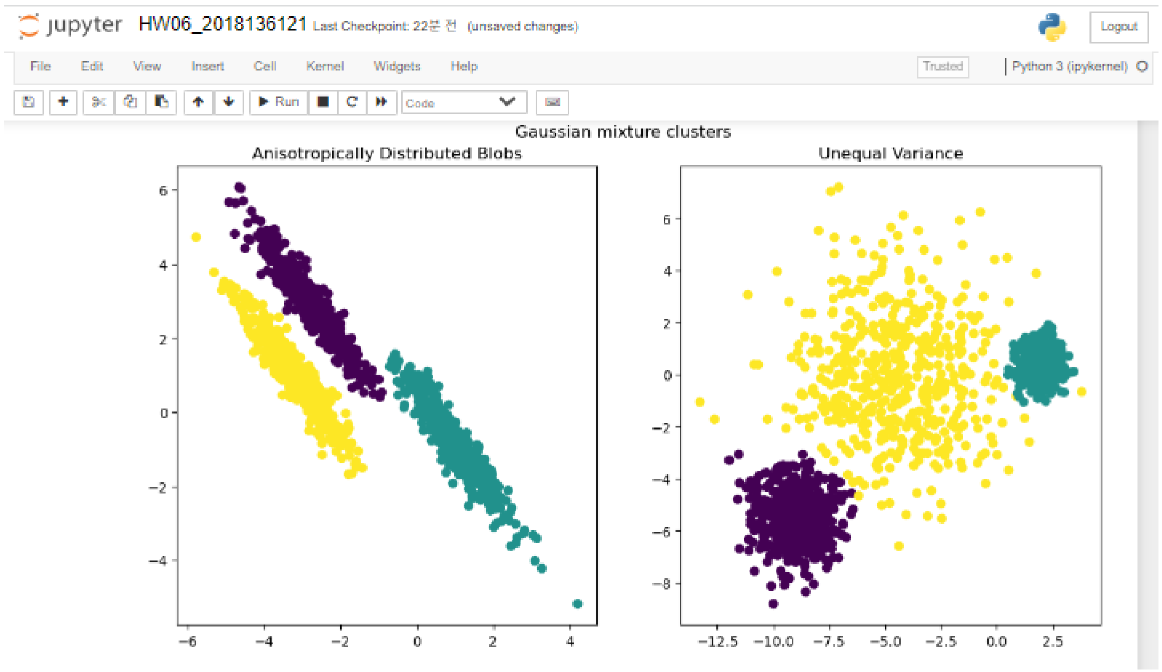

jupyter HW06_2018136121 Last Checkpoint: 21분 전 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Optimal Number of Clusters
2
0
−2
−4
−6
−8
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0
Unevenly Sized Blobs
with several initializations
2
0
−2
−4
−6
−8
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0

```
In [5]: y_pred = KMeans(n_clusters=3, **common_params).fit_predict(X)
        plt.scatter(X[:, 0], X[:, 1], c=y_pred)
        plt.title("Optimal Number of Clusters")
        plt.show()
```

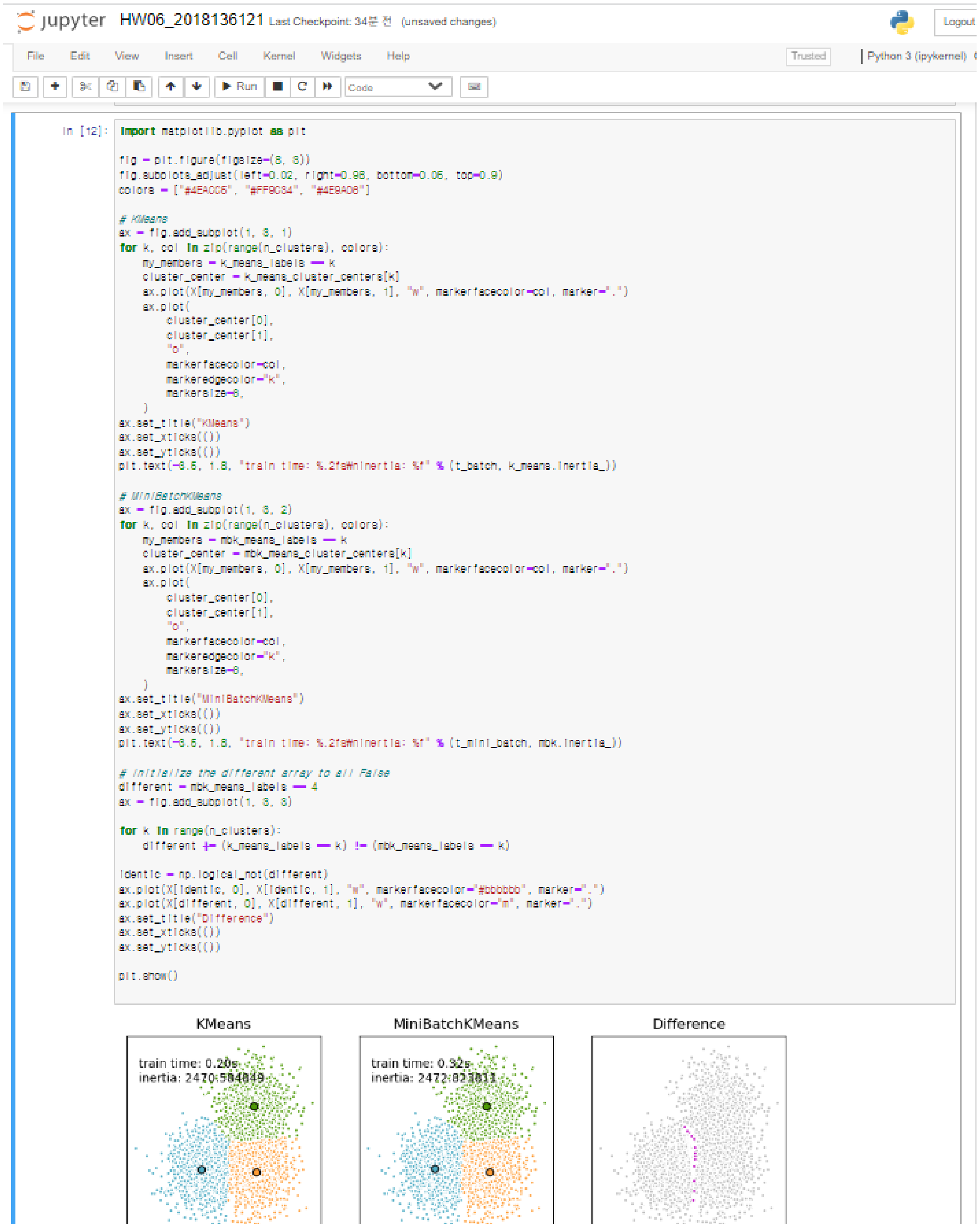
```
C:\Users\lgeq1\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=6.
  warnings.warn(
```

```
In [6]: y_pred = KMeans(n_clusters=3, n_init=10, random_state=random_state).fit_predict(
            X_filtered
        )
        plt.scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_pred)
        plt.title("Unevenly Sized Blobs \nwith several initializations")
        plt.show()
```

```
C:\Users\lgeq1\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```


jupyter HW06_2018136121 Last Checkpoint: 22분 전 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Gaussian mixture clusters
Anisotropically Distributed Blobs
6
4
2
0
−2
−4
−6 −4 −2 0 2 4
Unequal Variance
6
4
2
0
−2
−4
−6
−8
−12.5 −10.0 −7.5 −5.0 −2.5 0.0 2.5

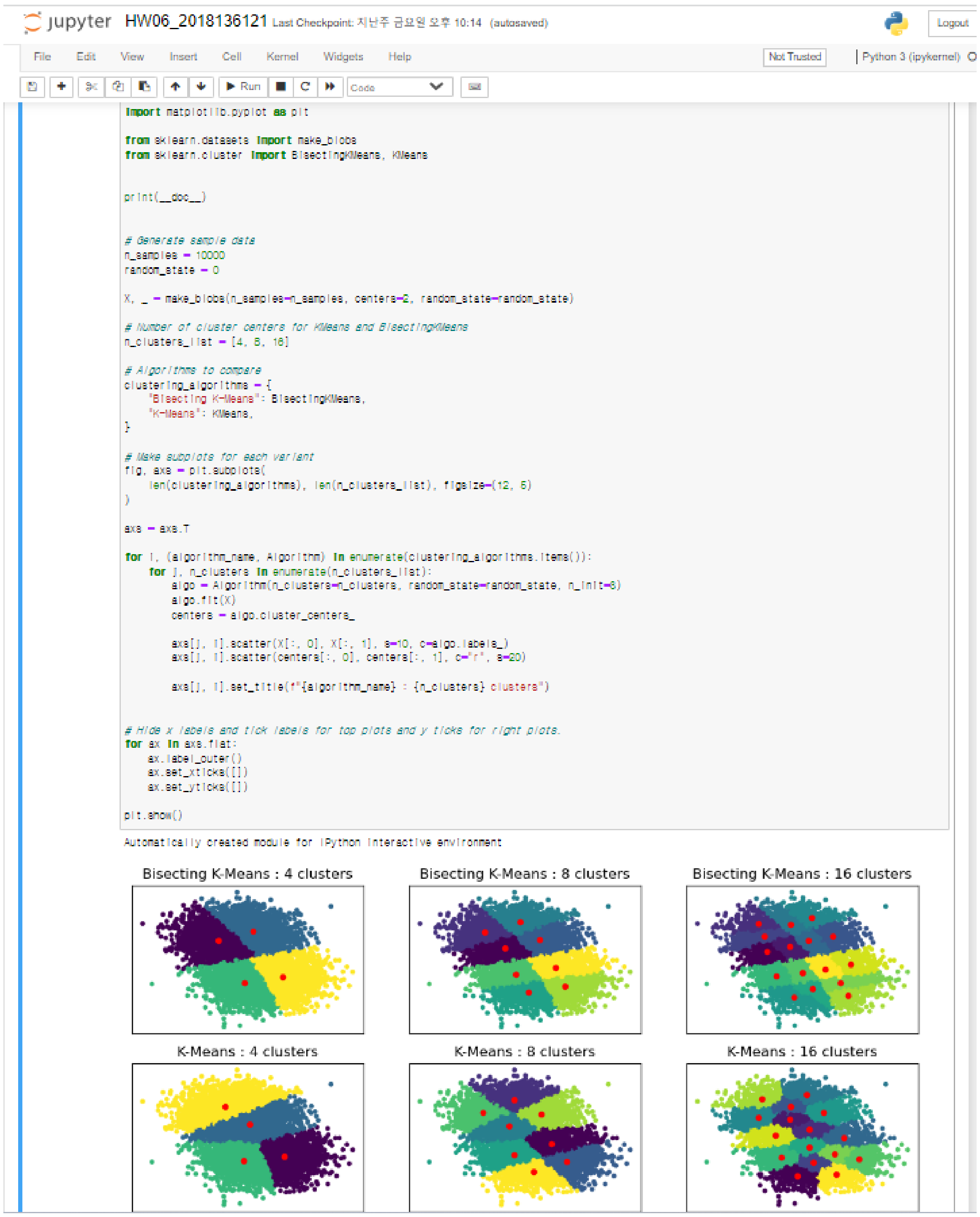
3. Comparison of the K-Means and MiniBatchKMeans clustering algorithms


jupyter HW06_2018136121 Last Checkpoint: 34분 전 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [12]: import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8, 3))
fig.subplots_adjust(left=0.02, right=0.98, bottom=0.05, top=0.9)
colors = ["#4EACC5", "#FF9C34", "#4E9A06"]

# KMeans
ax = fig.add_subplot(1, 3, 1)
for k, col in zip(range(n_clusters), colors):
    my_members = k_means_labels == k
    cluster_center = k_means_cluster_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], "w", markerfacecolor=col, marker=".")
    ax.plot(
        cluster_center[0],
        cluster_center[1],
        "o",
        markerfacecolor=col,
        markeredgecolor="k",
        markersize=6,
    )
ax.set_title("KMeans")
ax.set_xticks(())
ax.set_yticks(())
plt.text(-3.5, 1.8, "train time: %.2fs\ninertia: %f" % (t_batch, k_means.inertia_))

# MiniBatchKMeans
ax = fig.add_subplot(1, 3, 2)
for k, col in zip(range(n_clusters), colors):
    my_members = mbk_means_labels == k
    cluster_center = mbk_means_cluster_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], "w", markerfacecolor=col, marker=".")
    ax.plot(
        cluster_center[0],
        cluster_center[1],
        "o",
        markerfacecolor=col,
        markeredgecolor="k",
        markersize=6,
    )
ax.set_title("MiniBatchKMeans")
ax.set_xticks(())
ax.set_yticks(())
plt.text(-3.5, 1.8, "train time: %.2fs\ninertia: %f" % (t_mini_batch, mbk.inertia_))

# Initialize the different array to all False
different = mbk_means_labels == 4
ax = fig.add_subplot(1, 3, 3)

for k in range(n_clusters):
    different += (k_means_labels == k) != (mbk_means_labels == k)

identic = np.logical_not(different)
ax.plot(X[identic, 0], X[identic, 1], "w", markerfacecolor="#bbbbbb", marker=".")
ax.plot(X[different, 0], X[different, 1], "w", markerfacecolor="m", marker=".")
ax.set_title("Difference")
ax.set_xticks(())
ax.set_yticks(())

plt.show()
```

```
KMeans
train time: 0.20s
inertia: 2470.584849

MiniBatchKMeans
train time: 0.32s
inertia: 2472.823811

Difference
```

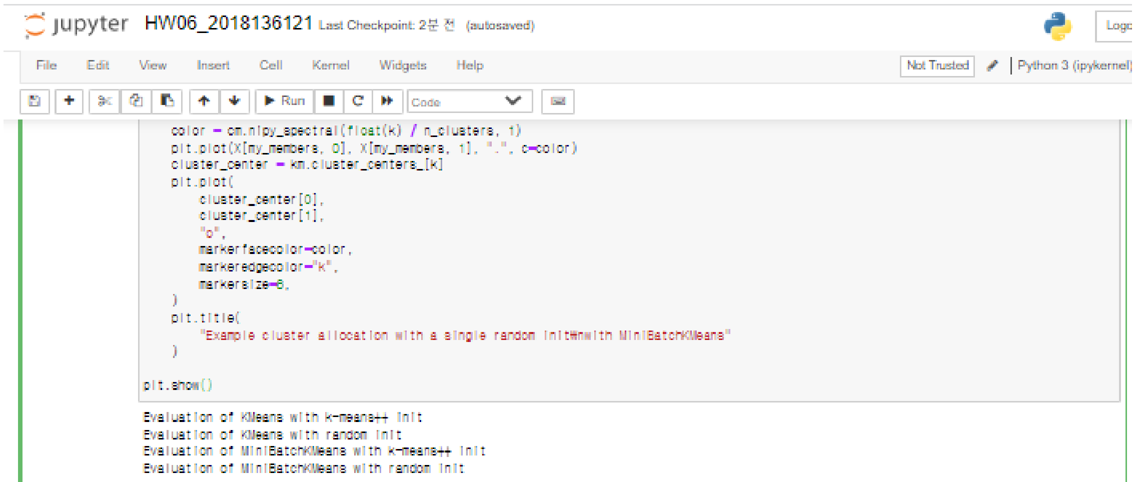
4. Bisecting K-Means and Regular K-Means Performance Comparison


jupyter HW06_2018136121 Last Checkpoint: 지난주 금요일 오후 10:14 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Bisecting K-Means : 4 clusters
Bisecting K-Means : 8 clusters
Bisecting K-Means : 16 clusters
K-Means : 4 clusters
K-Means : 8 clusters
K-Means : 16 clusters

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
from sklearn.cluster  import BisectingKMeans, KMeans


print(__doc__)


# Generate sample data
n_samples = 10000
random_state = 0

X, _ = make_blobs(n_samples=n_samples, centers=2, random_state=random_state)

# Number of cluster centers for KMeans and BisectingKMeans
n_clusters_list = [4, 8, 16]

# Algorithms to compare
clustering_algorithms = {
    "Bisecting K-Means": BisectingKMeans,
    "K-Means": KMeans,
}

# Make subplots for each variant
fig, axs = plt.subplots(
    len(clustering_algorithms), len(n_clusters_list), figsize=(12, 5)
)

axs = axs.T

for i, (algorithm_name, Algorithm) in enumerate(clustering_algorithms.items()):
    for j, n_clusters in enumerate(n_clusters_list):
        algo = Algorithm(n_clusters=n_clusters, random_state=random_state, n_init=3)
        algo.fit(X)
        centers = algo.cluster_centers_

        axs[j, i].scatter(X[:, 0], X[:, 1], s=10, c=algo.labels_)
        axs[j, i].scatter(centers[:, 0], centers[:, 1], c="r", s=20)

        axs[j, i].set_title(f"{algorithm_name} : {n_clusters} clusters")


# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
    ax.set_xticks([])
    ax.set_yticks([])

plt.show()
```

```
Automatically created module for IPython interactive environment
```

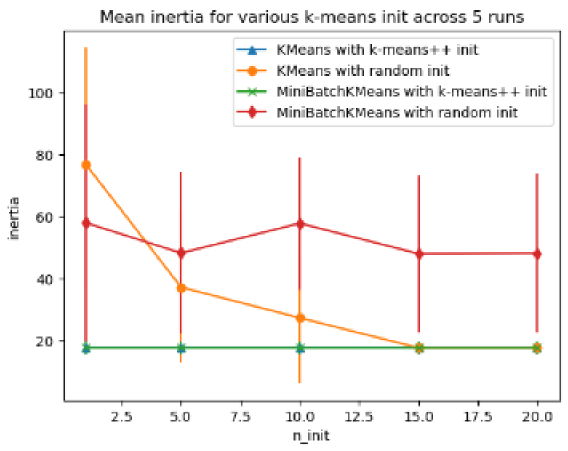
5. Empirical evaluation of the impact of k-means initialization

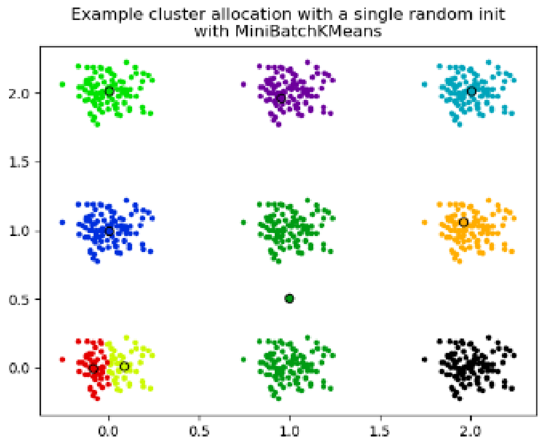


jupyter HW06_2018136121 Last Checkpoint: 2분 전 (autosaved)

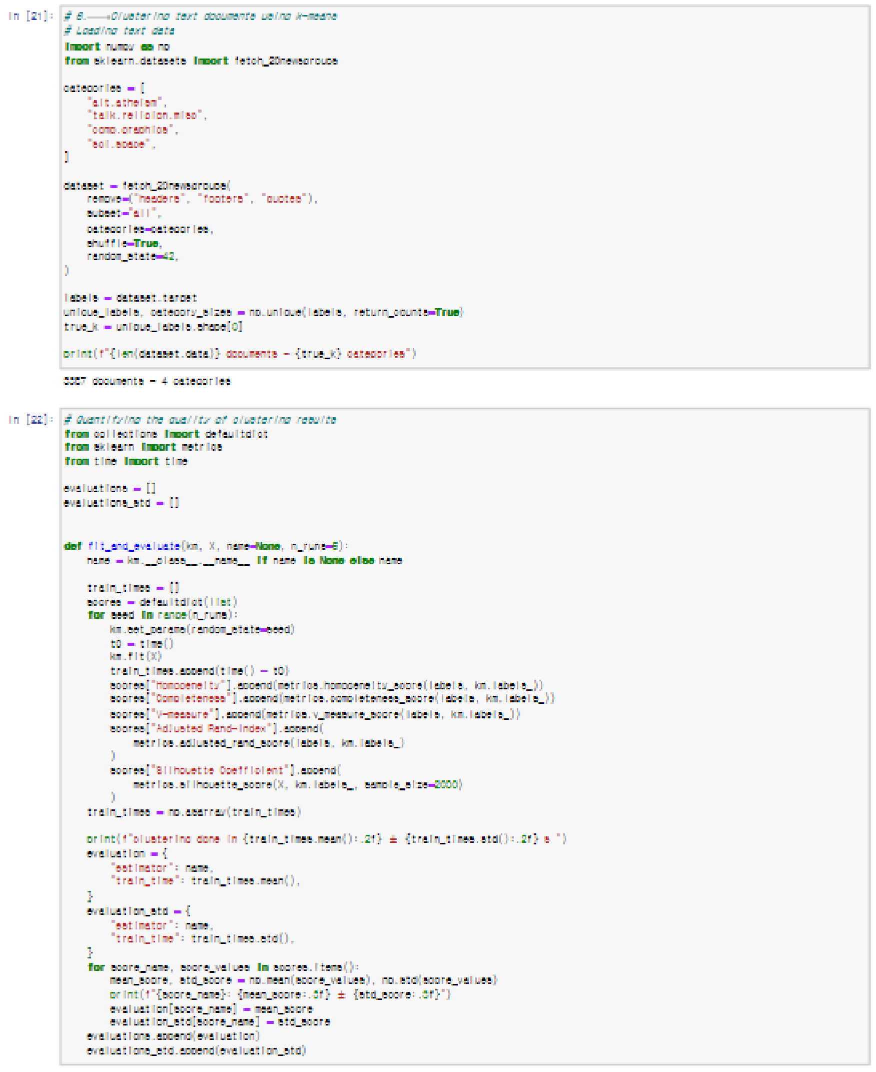
```
    color = cm.nipy_spectral(float(k) / n_clusters, 1)
    plt.plot(X[my_members, 0], X[my_members, 1], ".", c=color)
    cluster_center = km.cluster_centers_[k]
    plt.plot(
        cluster_center[0],
        cluster_center[1],
        "o",
        markerfacecolor=color,
        markeredgecolor="k",
        markersize=6,
    )
    plt.title(
        "Example cluster allocation with a single random init₩nwith MiniBatchKMeans"
    )

plt.show()
```

```
Evaluation of KMeans with k-means++ init
Evaluation of KMeans with random init
Evaluation of MiniBatchKMeans with k-means++ init
Evaluation of MiniBatchKMeans with random init
```





6. Clustering text documents using k-means



In [21]:
```python
# 6.——→Clustering text documents using k-means
# Loading text data
import numpy as np
from sklearn.datasets import fetch_20newsgroups

categories = [
    "alt.atheism",
    "talk.religion.misc",
    "comp.graphics",
    "sci.space",
]

dataset = fetch_20newsgroups(
    remove=("headers", "footers", "quotes"),
    subset="all",
    categories=categories,
    shuffle=True,
    random_state=42,
)

labels = dataset.target
unique_labels, category_sizes = np.unique(labels, return_counts=True)
true_k = unique_labels.shape[0]

print(f"{len(dataset.data)} documents - {true_k} categories")
```

```
3387 documents - 4 categories
```

In [22]:
```python
# Quantifying the quality of clustering results
from collections import defaultdict
from sklearn import metrics
from time import time

evaluations = []
evaluations_std = []


def fit_and_evaluate(km, X, name=None, n_runs=5):
    name = km.__class__.__name__ if name is None else name

    train_times = []
    scores = defaultdict(list)
    for seed in range(n_runs):
        km.set_params(random_state=seed)
        t0 = time()
        km.fit(X)
        train_times.append(time() - t0)
        scores["Homogeneity"].append(metrics.homogeneity_score(labels, km.labels_))
        scores["Completeness"].append(metrics.completeness_score(labels, km.labels_))
        scores["V-measure"].append(metrics.v_measure_score(labels, km.labels_))
        scores["Adjusted Rand-Index"].append(
            metrics.adjusted_rand_score(labels, km.labels_)
        )
        scores["Silhouette Coefficient"].append(
            metrics.silhouette_score(X, km.labels_, sample_size=2000)
        )
    train_times = np.asarray(train_times)

    print(f"clustering done in {train_times.mean():.2f} ± {train_times.std():.2f} s ")
    evaluation = {
        "estimator": name,
        "train_time": train_times.mean(),
    }
    evaluation_std = {
        "estimator": name,
        "train_time": train_times.std(),
    }
    for score_name, score_values in scores.items():
        mean_score, std_score = np.mean(score_values), np.std(score_values)
        print(f"{score_name}: {mean_score:.3f} ± {std_score:.3f}")
        evaluation[score_name] = mean_score
        evaluation_std[score_name] = std_score
    evaluations.append(evaluation)
    evaluations_std.append(evaluation_std)
```

In [24]:
```
print(f"{X_tfidf.nnz / np.prod(X_tfidf.shape):.3f}")
```

```
0.007
```

In [26]:
```
# Clustering sparse data with k-means
from sklearn.cluster import KMeans

for seed in range(5):
    kmeans = KMeans(
        n_clusters=true_k,
        max_iter=100,
        n_init=1,
        random_state=seed,
    ).fit(X_tfidf)
    cluster_ids, cluster_sizes = np.unique(kmeans.labels_, return_counts=True)
    print(f"Number of elements asigned to each cluster: {cluster_sizes}")
print()
print(
    "True number of documents in each category according to the class labels: "
    f"{category_sizes}"
)
```

```
Number of elements asigned to each cluster: [   1    1 3384    1]
Number of elements asigned to each cluster: [1597  732  233  826]
Number of elements asigned to each cluster: [2004  448  646  291]
Number of elements asigned to each cluster: [1695  649  446  597]
Number of elements asigned to each cluster: [ 254 2117  459  557]

True number of documents in each category according to the class labels: [799 973 987 628]
```

In [27]:
```
#To avoid this problem, one possibility is to increase the number of runs with independent
# random initiations n_init. In such case the clustering with the best inertia (objective function of k-means) is chosen.

kmeans = KMeans(
    n_clusters=true_k,
    max_iter=100,
    n_init=5,
)

fit_and_evaluate(kmeans, X_tfidf, name="KMeans₩non tf-idf vectors")
```

```
clustering done in 1.16 ± 0.10 s
Homogeneity: 0.338 ± 0.028
Completeness: 0.400 ± 0.008
V-measure: 0.365 ± 0.014
Adjusted Rand-Index: 0.202 ± 0.012
Silhouette Coefficient: 0.008 ± 0.001
```

In [28]:
```
# Performing dimensionality reduction using LSA
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer


lsa = make_pipeline(TruncatedSVD(n_components=100), Normalizer(copy=False))
t0 = time()
X_lsa = lsa.fit_transform(X_tfidf)
explained_variance = lsa[0].explained_variance_ratio_.sum()

print(f"LSA done in {time() - t0:.3f} s")
print(f"Explained variance of the SVD step: {explained_variance * 100:.1f}%")
```

```
LSA done in 1.019 s
Explained variance of the SVD step: 18.4%
```

In [29]:
```
#Using a single initialization means the processing time will be reduced for both KMeans and MiniBatchKMeans.

kmeans = KMeans(
    n_clusters=true_k,
    max_iter=100,
    n_init=1,
)

fit_and_evaluate(kmeans, X_lsa, name="KMeans₩nwith LSA on tf-idf vectors")
```

```
clustering done in 0.11 ± 0.01 s
Homogeneity: 0.368 ± 0.044
Completeness: 0.417 ± 0.021
V-measure: 0.401 ± 0.034
Adjusted Rand-Index: 0.303 ± 0.062
Silhouette Coefficient: 0.029 ± 0.002
```

```
In [32]: #We repeat the experiment with MiniBatchKMeans.

         from sklearn.cluster import MiniBatchKMeans

         minibatch_kmeans = MiniBatchKMeans(
             n_clusters=true_k,
             n_init=1,
             init_size=1000,
             batch_size=1000,
         )

         fit_and_evaluate(
             minibatch_kmeans,
             X_lsa,
             name="MiniBatchKMeans₩nwith LSA on tf-idf vectors",
         )

         clustering done in 0.52 ± 0.08 s
         Homogeneity: 0.392 ± 0.027
         Completeness: 0.399 ± 0.018
         V-measure: 0.395 ± 0.021
         Adjusted Rand-Index: 0.348 ± 0.029
         Silhouette Coefficient: 0.028 ± 0.001
```

```
In [33]: # Top terms per cluster
         original_space_centroids = lsa[0].inverse_transform(kmeans.cluster_centers_)
         order_centroids = original_space_centroids.argsort()[:, ::-1]
         terms = vectorizer.get_feature_names_out()

         for i in range(true_k):
             print(f"Cluster {i}: ", end="")
             for ind in order_centroids[i, :10]:
                 print(f"{terms[ind]} ", end="")
             print()

         Cluster 0: graphics software computer comp edu information image like available new
         Cluster 1: god jesus bible people christian believe don say religion christians
         Cluster 2: thanks files file program image know help looking format does
         Cluster 3: space just think don people like know time say way
```

```
In [34]: # HashingVectorizer
         from sklearn.feature_extraction.text import HashingVectorizer
         from sklearn.feature_extraction.text import TfidfTransformer

         lsa_vectorizer = make_pipeline(
             HashingVectorizer(stop_words="english", n_features=50_000),
             TfidfTransformer(),
             TruncatedSVD(n_components=100, random_state=0),
             Normalizer(copy=False),
         )

         t0 = time()
         X_hashed_lsa = lsa_vectorizer.fit_transform(dataset.data)
         print(f"vectorization done in {time() - t0:.3f} s")

         vectorization done in 3.919 s
```

```
In [35]: # We now fit and evaluate the kmeans and minibatch_kmeans instances on this hashed-lsa-reduced data:

         fit_and_evaluate(kmeans, X_hashed_lsa, name="KMeans₩nwith LSA on hashed vectors")

         clustering done in 0.12 ± 0.02 s
         Homogeneity: 0.393 ± 0.010
         Completeness: 0.434 ± 0.024
         V-measure: 0.412 ± 0.015
         Adjusted Rand-Index: 0.331 ± 0.018
         Silhouette Coefficient: 0.028 ± 0.004
```

```
In [36]: fit_and_evaluate(
             minibatch_kmeans,
             X_hashed_lsa,
             name="MiniBatchKMeans₩nwith LSA on hashed vectors",
         )

         clustering done in 0.50 ± 0.07 s
         Homogeneity: 0.339 ± 0.054
         Completeness: 0.349 ± 0.051
         V-measure: 0.344 ± 0.053
         Adjusted Rand-Index: 0.307 ± 0.060
         Silhouette Coefficient: 0.026 ± 0.003
```

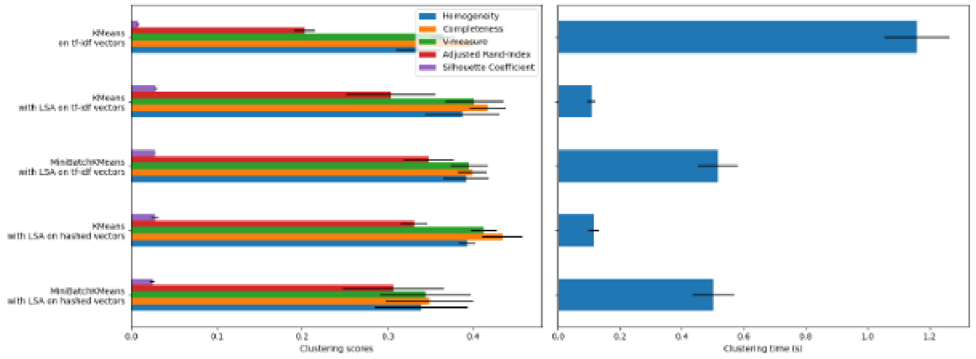
```
In [35]: # We now fit and evaluate the kmeans and minibatch_kmeans instances on this hashed-lsa-reduced data:

         fit_and_evaluate(kmeans, X_hashed_lsa, name="KMeans₩nwith LSA on hashed vectors")
```

```
clustering done in 0.12 ± 0.02 s
Homogeneity: 0.393 ± 0.010
Completeness: 0.434 ± 0.024
V-measure: 0.412 ± 0.015
Adjusted Rand-Index: 0.331 ± 0.016
Silhouette Coefficient: 0.028 ± 0.004
```

```
In [36]: fit_and_evaluate(
             minibatch_kmeans,
             X_hashed_lsa,
             name="MiniBatchKMeans₩nwith LSA on hashed vectors",
         )
```

```
clustering done in 0.50 ± 0.07 s
Homogeneity: 0.339 ± 0.054
Completeness: 0.349 ± 0.051
V-measure: 0.344 ± 0.053
Adjusted Rand-Index: 0.307 ± 0.060
Silhouette Coefficient: 0.026 ± 0.003
```

```
In [37]: # Clustering evaluation summary
         import pandas as pd
         import matplotlib.pyplot as plt

         fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(16, 6), sharey=True)

         df = pd.DataFrame(evaluations[::-1]).set_index("estimator")
         df_std = pd.DataFrame(evaluations_std[::-1]).set_index("estimator")

         df.drop(
             ["train_time"],
             axis="columns",
         ).plot.barh(ax=ax0, xerr=df_std)
         ax0.set_xlabel("Clustering scores")
         ax0.set_ylabel("")

         df["train_time"].plot.barh(ax=ax1, xerr=df_std["train_time"])
         ax1.set_xlabel("Clustering time (s)")
         plt.tight_layout()
```


Homogeneity
Completeness
V-measure
Adjusted Rand-Index
Silhouette Coefficient
KMeans
on tf-idf vectors
KMeans
with LSA on tf-idf vectors
MiniBatchKMeans
with LSA on tf-idf vectors
KMeans
with LSA on hashed vectors
MiniBatchKMeans
with LSA on hashed vectors
Clustering scores
Clustering time (s)