hw3-unsupervised

June 4, 2025

1 hw3:

mlcourse.ai : (@aiho Slack ODS), (@yorko Slack ODS).

1.0.1

, . .

Samsung Human Activity Recognition.
Samsung Galaxy S3 (-- UCI),
-- , , , / .

, , .
.

(`` '') , (** **).

1.0.2

10 2 . « » ( ).
--- 10 . / .

1.0.3

`hw3-unsupervised.ipynb` Github. Google-
``hw3'' , .

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm_notebook

%matplotlib inline
from matplotlib import pyplot as plt
import seaborn as sns

sns.set_style('darkgrid')
plt.rcParams['figure.figsize'] = (12, 9)
plt.rcParams['font.family'] = 'DejaVu Sans'
```

```python
from sklearn import metrics
from sklearn.cluster import AgglomerativeClustering, KMeans, SpectralClustering
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

RANDOM_STATE = 17
```

[2]:
```python
import os
from pathlib import Path

print("Working dir:", os.getcwd())

base = Path.cwd() / "datasets" / "human+activity+recognition+using+smartphones"␣
 ↪\
       / "UCI HAR Dataset" / "UCI HAR Dataset"

train_dir = base / "train"
test_dir = base / "test"
```

```
Working dir: /home/sergey/PycharmProjects/ML_UNI/HW_3
```

[3]:
```python
X_train = np.loadtxt(train_dir / "X_train.txt")
y_train = np.loadtxt(train_dir / "y_train.txt", dtype=int)
subject_train = np.loadtxt(train_dir / "subject_train.txt", dtype=int)

X_test = np.loadtxt(test_dir / "X_test.txt")
y_test = np.loadtxt(test_dir / "y_test.txt", dtype=int)
subject_test = np.loadtxt(test_dir / "subject_test.txt", dtype=int)

print("Loaded:", X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
Loaded: (7352, 561) (7352,) (2947, 561) (2947,)
```

[4]:
```python
#
assert (X_train.shape == (7352, 561) and y_train.shape == (7352,))
assert (X_test.shape == (2947, 561) and y_test.shape == (2947,))
```

, . *X_train* *X_test*, *y_train* -- *y_test*.

[5]:
```python
#
X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate([y_train, y_test], axis=0)
```

.

[6]:
```python
np.unique(y)
```

[6]: array([1, 2, 3, 4, 5, 6])

[7]:
```
n_classes = np.unique(y).size
```

: - 1 - - 2 - - 3 - - 4 - - 5 - - 6 -

, :)

StandardScaler .

[8]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("mean per feature:", np.round(X_scaled.mean(axis=0)[:5], 3), "…")
print("std  per feature:", np.round(X_scaled.std(axis=0)[:5], 3), "…")
```

```
mean per feature: [ 0. -0. -0. -0.  0.] …
std  per feature: [1. 1. 1. 1. 1.] …
```

PCA, , , 90% ( ) . random_state (RANDOM_STATE).

[9]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.9, random_state=RANDOM_STATE)
X_pca = pca.fit_transform(X_scaled)

print("            :", pca.explained_variance_ratio_.sum())
print("         :", pca.n_components_)
```

```
: 0.9004833346822924
: 65
```

1: (1)

, 90% ( ) ?

: - 56 - **65** - 66 - 193

[10]:
```
print("         :", pca.n_components_)
```

```
: 65
```

2: (0.5)

? .

: - 45 - **51** - 56 - 61

[11]:
```
first_ratio = pca.explained_variance_ratio_[0]

print(round(first_ratio * 100))
```

```
51
```

.

```
[12]: plt.figure(figsize=(12, 9))
plt.scatter(X_pca[:, 0],
            X_pca[:, 1],
            c=y,
            s=20,
            cmap='viridis');
plt.xlabel('PC1 ')
plt.ylabel('PC2')
plt.colorbar(label='True labels')
plt.show()
```

3: (0.5)

, - , . ?

: - 1 : 6 - 2 : (, ,) (, ,) - 3 : ( ), (,) (, ,) **- 6**

---

1. (1)

KMeans ( ),
PCA . , 6 , , .

:

- **n_clusters** = n_classes ( )
- **n_init** = 100
- **random_state** = RANDOM_STATE ( )

.

[13]:
```python
from sklearn.cluster import KMeans

k_means = KMeans(
    n_clusters=n_classes,
    n_init=100,
    random_state=RANDOM_STATE
)
cluster_labels = k_means.fit_predict(X_pca)
centroids_sklearn = k_means.cluster_centers_

print("Inertia:", k_means.inertia_)
```

```
Inertia: 2003454.8999158153
```

[14]:
```python
import numpy as np


def kmeansManual(X, n_clusters, n_init=10, max_iter=300, tol=1e-4):
    best_inertia = np.inf
    best_labels = None
    best_centroids = None

    for init_no in range(n_init):
        index = np.random.choice(X.shape[0], n_clusters, replace=False)
        centroids = X[index].copy()

        for iteration in range(max_iter):
            distances = np.linalg.norm(X[:, None] - centroids[None, :], axis=2)
            labels = np.argmin(distances, axis=1)

            new_centorids = np.array([
                X[labels == k].mean(axis=0) if np.any(labels == k) else centroids[k]
                for k in range(n_clusters)
            ])
```

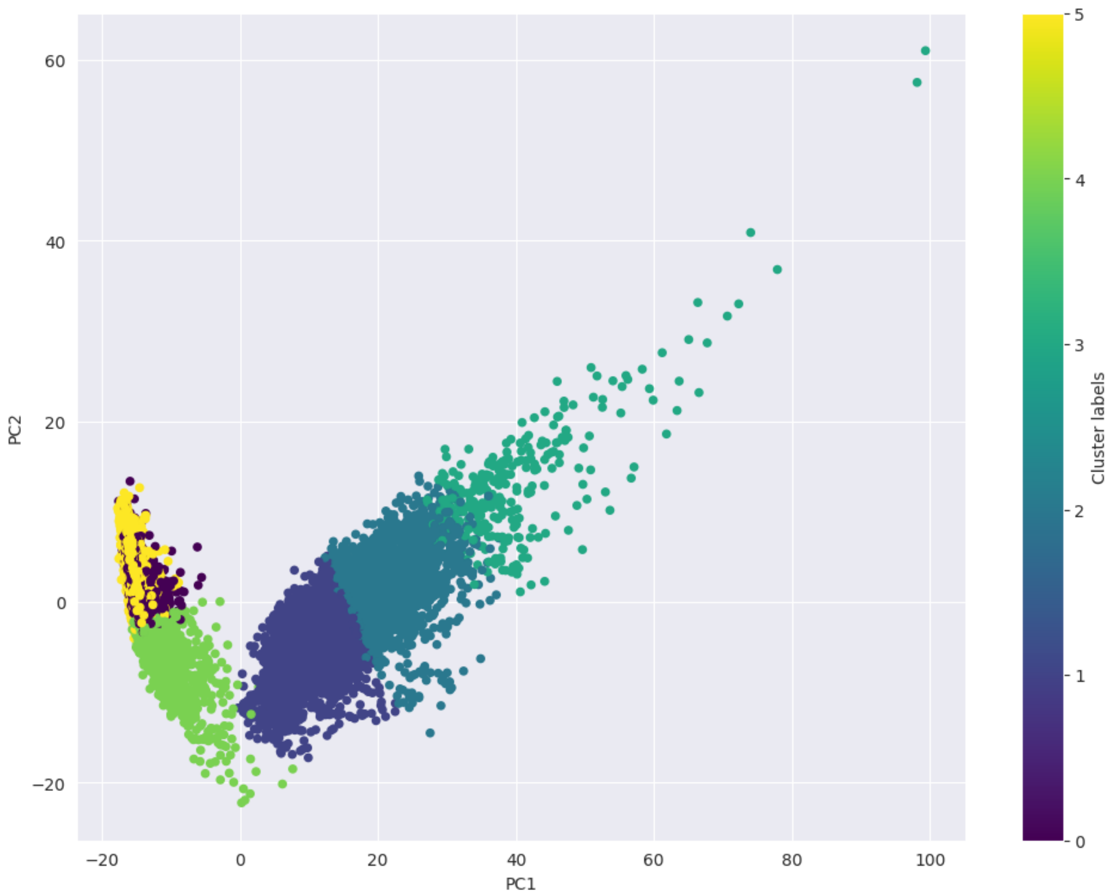
```
            shift = np.linalg.norm(new_centorids - centroids, axis=1).max()
            centroids = new_centorids
            if shift < tol:
                break
        inertia = sum(((X[labels == k] - centroids[k]) ** 2).sum() for k in␣
 ↪range(n_clusters))

        if inertia < best_inertia:
            best_inertia = inertia
            best_labels = labels.copy()
            best_centroids = centroids.copy()
    return best_labels, best_centroids, best_inertia


cluster_labels_custom, centroids_custom, inertia_custom = kmeansManual(
    X_pca,
    n_clusters=n_classes,
    n_init=100,
    max_iter=300,
    tol=1e-4
)

print("KMeans inertia:", inertia_custom)
```

```
KMeans inertia: 2003454.7982992886
```

. .

[15]:

```
plt.figure(figsize=(12, 9))
plt.scatter(
    X_pca[:, 0],
    X_pca[:, 1],
    c=cluster_labels,
    s=20,
    cmap='viridis');
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label='Cluster labels')
plt.show()
```



, KMeans .

[16]:
```python
tab = pd.crosstab(y, cluster_labels, margins=True)
tab.index = ['    ', '                  ',
             '          ', '    ', '    ', '    ', '  ']
tab.columns = ['cluster' + str(i + 1) for i in range(6)] + ['  ']
tab
```

[16]:

cluster1	cluster2	cluster3	cluster4	cluster5	\
0	903	741	78	0	
0	1241	296	5	2	
0	320	890	196	0	
1235	1	0	0	450	
1344	0	0	0	562	
52	5	0	0	329	
2631	2470	1927	279	1343	

cluster6	
0	1722

```
0    1544
   0    1406
      91    1777
        0    1906
   1558    1944
      1649   10299
```

, (. .) . ,
- . , , .

: `` '', 1406 , : - 1 -- 900 - 3 --
500 - 6 -- 6,

900 / 1406 ≈ 0.64.

4: (1)

, ?

: - - - -

[17]:
```python
cluster_counts = tab.iloc[:-1, :-1]

total_per_class = tab['  '][:-1]

best_frac_per_class = cluster_counts.div(total_per_class, axis=0).max(axis=1)

best_activity = best_frac_per_class.idxmax()
best_value    = best_frac_per_class.max()

print(f"Best activity: «{best_activity}» with percent {best_value:.2%}")
```

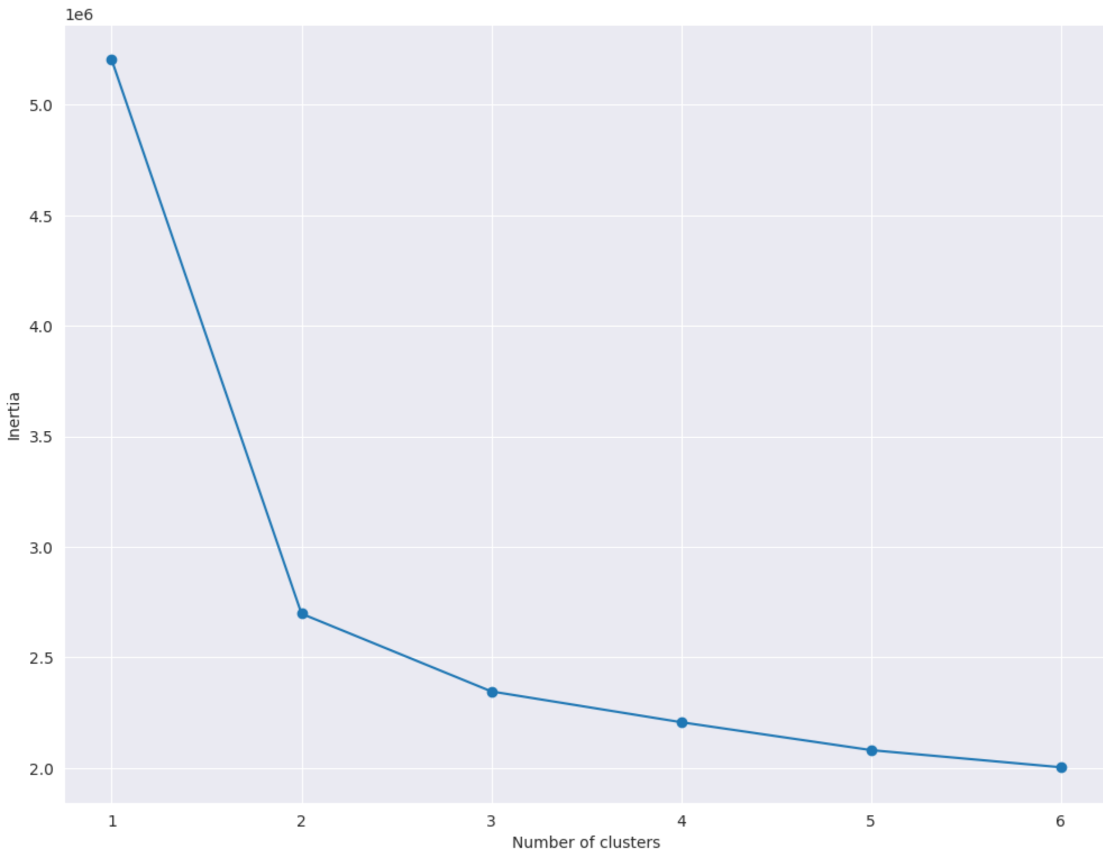
```
Best activity: «                » with percent 80.38%
```

, kMeans . ,
. , , `n_clusters`.

[18]:
```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from tqdm import tqdm

inertia = []
ks = range(1, n_classes + 1)
for k in tqdm(ks):
    kmeans = KMeans(
        n_clusters=k,
        n_init=100,
        random_state=RANDOM_STATE
    )
    kmeans.fit(X_pca)
    inertia.append(kmeans.inertia_)
```

```
plt.figure(figsize=(12, 9))
plt.plot(ks, inertia, marker='o')
plt.xticks(ks)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

```
100%|          | 6/6 [01:31<00:00, 15.26s/it]
```



5: (1)

, ?

: - 1 - **2** - 3 - 4

[19]:
```
xs = np.array(ks)
ys = np.array(inertia)

p1 = np.array([xs[0], ys[0]])
p2 = np.array([xs[-1], ys[-1]])
```

```
v = p2 - p1

distances = np.abs(
    v[1] * (xs - p1[0]) -
    v[0] * (ys - p1[1])
) / np.linalg.norm(v)

opt_k = xs[np.argmax(distances)]
print(f"Optimal: {opt_k}")
```

```
Optimal: 2
```

, -- .

[20]:
```
ag = AgglomerativeClustering(n_clusters=n_classes,
                             linkage='ward').fit(X_pca)
labels_ag = ag.labels_
```

Adjusted Rand Index (`sklearn.metrics`) `KMeans`
4 .

[21]:
```
from sklearn.metrics import adjusted_rand_score

kmeans = KMeans(
    n_clusters=n_classes,
    n_init=100,
    random_state=RANDOM_STATE
).fit(X_pca)
labels_km = kmeans.labels_

ari_ag = adjusted_rand_score(y, labels_ag)
ari_km = adjusted_rand_score(y, labels_km)
print(f"Agglomerative: {ari_ag:.4f}")
print(f"Means:         {ari_km:.4f}")
```

```
Agglomerative: 0.4936
Means:         0.4198
```

6: (1)

.

: - ARI, KMeans , Agglomerative Clustering - ARI
, -
ARI

, , ($>$ 2).
, , .

-- sklearn.svm.LinearSVC.
, , , , -- .

LinearSVC C GridSearchCV.

- StandardScaler ( ),
- GridSearchCV cv=3.

[22]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[23]:
```
svc = LinearSVC(random_state=RANDOM_STATE)
svc_params = {'C': [0.001, 0.01, 0.1, 1, 10]}
```

[24]:
```
grid = GridSearchCV(
    estimator=svc,
    param_grid=svc_params,
    cv=3,
    n_jobs=-1,
)
grid.fit(X_train_scaled, y_train)
best_svc = grid.best_estimator_
best_score_orig = grid.best_score_
```

```
/home/sergey/PycharmProjects/ML_UNI/.venv/lib/python3.13/site-
packages/sklearn/svm/_base.py:1249: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn(
/home/sergey/PycharmProjects/ML_UNI/.venv/lib/python3.13/site-
packages/sklearn/svm/_base.py:1249: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn(
/home/sergey/PycharmProjects/ML_UNI/.venv/lib/python3.13/site-
packages/sklearn/svm/_base.py:1249: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn(
```

[25]:
```
print("Best params:", grid.best_params_)
print("CV-accuracy on train:", grid.best_score_)
print("Accuracy on test:    ", best_svc.score(X_test_scaled, y_test))
```

```
Best params: {'C': 0.1}
CV-accuracy on train: 0.9379785010699506
Accuracy on test:    0.9619952494061758
```

7 (0.5)

C - ?

: - 0.001 - 0.01 - **0.1** - 1 - 10

[26]:
```python
y_predicted = best_svc.predict(X_test_scaled)
```

[27]:
```python
tab = pd.crosstab(y_test, y_predicted, margins=True)
tab.index = ['    ', '              ', '            ',
             '    ', '    ', '    ', '  ']
tab.columns = tab.index
tab
```

[27]:

				\
	494	2	0	
	12	459	0	
	2	4	413	
	0	4	0	
	0	0	0	
	0	0	0	
	508	469	413	

	0	0	0	496
	0	0	0	471
	1	0	0	420
	426	61	0	491
	15	517	0	532
	0	11	526	537
	442	589	526	2947

8: (0.5)

SVM ? ?

: - -- , -- - -- , -- - -- , -- - -- , --

-

[28]:
```python
from sklearn.metrics import precision_score, recall_score
import numpy as np

activity_names = [
    '    ',
    '              ',
    '            ',
    '    ',
    '    ',
    '    '
```

```
]

precision = precision_score(y_test, y_predicted, labels=np.arange(1, 7),␣
↪average=None)
recall    = recall_score(   y_test, y_predicted, labels=np.arange(1, 7),␣
↪average=None)

worst_prec_idx = np.argmin(precision)
worst_rec_idx  = np.argmin(recall)

print(f"        - {activity_names[worst_prec_idx]}")
print(f"          - {activity_names[worst_rec_idx]}")
```

```
       -
         -
```

, , 7 , PCA.

- X_train_scaled X_test_scaled
- PCA, , ,
- C - PCA- . , , .

9: (1)

( ) - 561 , ? .

: - - 2% - **4%** - 10% - 20%

[29]:
```
from sklearn.decomposition import PCA
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

pca = PCA(n_components=0.9, random_state=RANDOM_STATE)
Xtrain_pca = pca.fit_transform(X_train_scaled)
Xtest_pca = pca.transform(X_test_scaled)

svc = LinearSVC(random_state=RANDOM_STATE, max_iter=1000)
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}

grid = GridSearchCV(
    estimator=svc,
    param_grid=param_grid,
    cv=3,
    n_jobs=-1
)

grid.fit(Xtrain_pca, y_train)
```

```python
best_score_pca = grid.best_score_
best_svc_pca = grid.best_estimator_

diff_percent = round((best_score_orig - best_score_pca) * 100)
print(f"  Best C:        {grid.best_params_['C']}")
print(f"  CV-accuracy:     {grid.best_score_:.4f}")
print(f"  Test accuracy:   {best_svc_pca.score(Xtest_pca, y_test):.4f}")
print(f"Difference in quality CV: {diff_percent}%")
```

```
  Best C:        0.1
  CV-accuracy:     0.8984
  Test accuracy:   0.9192
Difference in quality CV: 4%
```

10: (1)

:

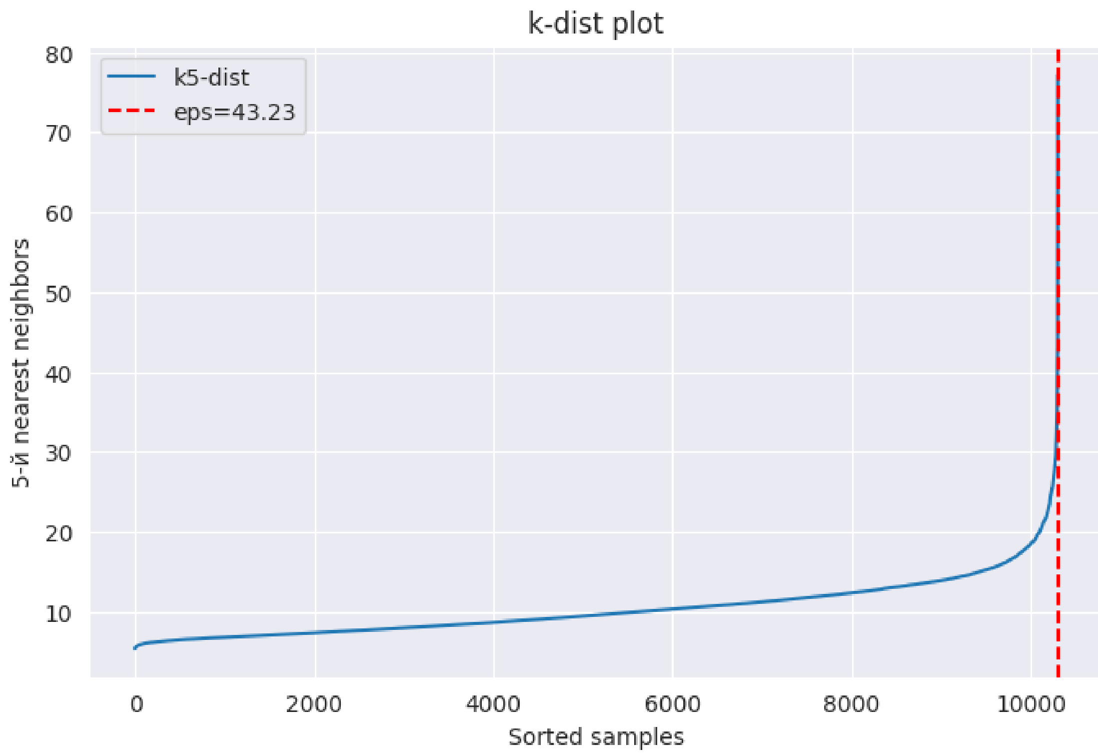
: - , (-)
, 10% - **PCA** , ,
, tSNE. PCA - PCA ,

2. (1)

DBSCAN tSNE.

[36]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

nn = NearestNeighbors(n_neighbors=5).fit(X_pca)
dists, _ = nn.kneighbors(X_pca)
k5 = np.sort(dists[:, 4])

d1 = np.gradient(k5)
d2 = np.gradient(d1)
idx_elbow = np.argmax(d2)
eps_choice = k5[idx_elbow]
print(f"chosen eps (k-dist): {eps_choice:.2f}")

plt.figure(figsize=(8,5))
plt.plot(k5, label='k5-dist')
plt.axvline(idx_elbow, color='red', linestyle='--', label=f'eps={eps_choice:.
 ↪2f}')
plt.legend()
plt.title('k-dist plot')
plt.ylabel('5-  nearest neighbors')
plt.xlabel('Sorted samples')
plt.show()
```

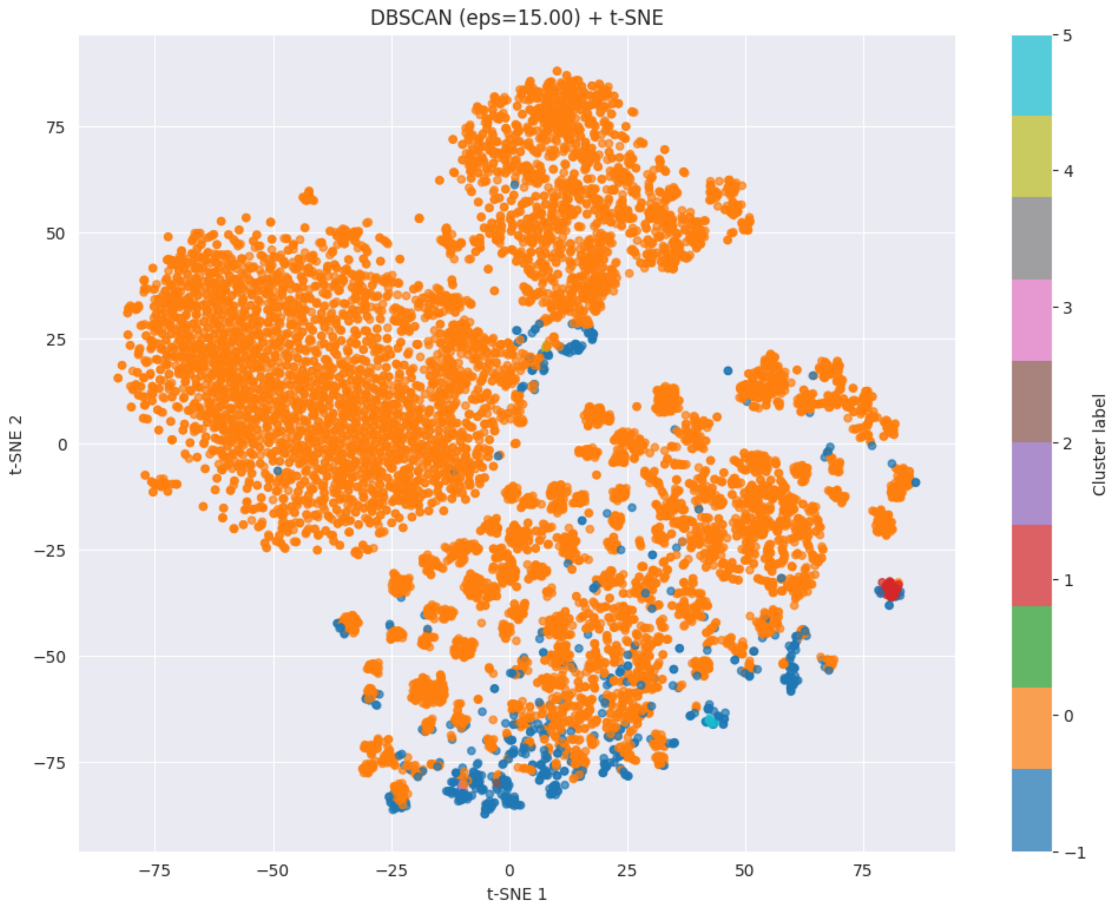
```
chosen eps (k-dist): 43.23
```



[43]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.manifold import TSNE
from collections import Counter

db = DBSCAN(eps=15, min_samples=5).fit(X_pca)
print("DBSCAN cluster counts:", Counter(db.labels_))

X_tsne = TSNE(n_components=2, random_state=RANDOM_STATE, init='random').
    ↪fit_transform(X_pca)

plt.figure(figsize=(12, 9))
plt.scatter(
    X_tsne[:, 0],
    X_tsne[:, 1],
    c=db.labels_,
    cmap='tab10',
    s=20,
    alpha=0.7
)
```

```
plt.colorbar(label='Cluster label')
plt.title(f'DBSCAN (eps={15:.2f}) + t-SNE')
plt.xlabel('t-SNE 1')
plt.ylabel('t-SNE 2')
plt.show()
```

```
DBSCAN cluster counts: Counter({np.int64(0): 9661, np.int64(-1): 573,
np.int64(1): 45, np.int64(5): 11, np.int64(4): 4, np.int64(3): 3, np.int64(2):
2})
```



epsilon