سروش پسندیده – ۹۹۱۲۷۶۲۱۰۹، سروش فتحی – ۹۹۱۲۷۶۲۷۳۴

در این فاز از ما خواسته شده که با استفاده از ویژگیهای (features) تهیه شده از تصاویر دیتاست flower102، ابتدا به صورت بدون ناظر (unsupervised) با الگوریتم k-means دیتاها را به صورت مناسب طبقه بندی کرده و سپس در بخش بعد با شکستن داده ها به گروه های کوچکتر و الگوریتم k-Nearest neighbors بهترین امتیاز را بدست آوریم.

بررسی کد

ابتدا کتابخانه های مورد نیاز را ایمپورت میکنیم:

```
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.manifold import TSNE
import pickle, os
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from typing import Dict, List, Tuple
from scipy.special import comb
```

با استفاده از تابع زیر داده ها را از فایل dataset-flowers102-features.pkl بارگزاری میکنیم:

```
def load_dataset(name_file):

    desired_directory = '.' # Replace with your desired directory path

    file_path = os.path.join(desired_directory, name_file)

    with open(file_path, 'rb') as f:
        data_dict = pickle.load(f)

    return data_dict
```

سپس داده های لود شده را به دو بخش test و train تقسیم میکنیم:

```
loaders_dict = load_dataset(f"dataset-flowers102-features.pkl")
x_train = loaders_dict["x_train"]
x_test = loaders_dict["x_test"]
y_train = loaders_dict["y_train"]
y_test = loaders_dict["y_test"]
```

```
x_train:(4094, 512), y_train:(4094,)
x_test:(4095, 512), y_test:(4095,)
```

در بخش 'clustering images' کد با استفاده از الگوریتم k-means با k=50، داده ها را کلاس بندی میکنیم:

```
k_number = 50
kmeans = KMeans(n_clusters=k_number)
clusters = kmeans.fit_predict(x_train)
centroids = kmeans.cluster_centers_ # (50, 512) <class 'numpy.ndarray'>
```

از متریک rand-index برای ارزیابی کیفیت کلاســـتر بندی الگوریتم k-means اســـتفاده میکنیم. فرمول محاسبه آن به صورت روبرو است که در تابع زیر پیاده سازی شده است:

RI= (TP+TN) / (TP+FP+FN+TN)

```
def rand_index_score(clusters, classes) -> float:
    """
    Calculate the Rand Index for two clusterings.

    The Rand Index is a measure of the similarity between two data clusterings.
    It's calculated based on the number of pairs of elements that are in the same
    cluster in both the predicted and actual clusterings, and the number of pairs
    of elements that are in different clusters in both the predicted and actual clusterings.

    Parameters:
    clusters (array-like): The predicted cluster labels.
    classes (array-like): The actual class labels.

    Returns:
    float: The Rand Index score, a value between 0 and 1. A higher value indicates a better
    agreement between the predicted and actual clusters.
    """
    tp_plus_fp = comb(np.bincount(clusters), 2).sum()
    tp_plus_fn = comb(np.bincount(classes), 2).sum()
    A = np.c_[(clusters, classes)]
    tp = sum(comb(np.bincount(A[A[:, 0] == i, 1]), 2).sum()
             for i in set(clusters))
    fp = tp_plus_fp - tp
    fn = tp_plus_fn - tp
    tn = comb(len(A), 2) - tp - fp - fn
    return (tp + tn) / (tp + fp + fn + tn)

rand_index = rand_index_score(clusters, y_train)
print('Rand Index:', rand_index)
```

```
Rand Index: 0.979636256260316
```

خروجی کد نشـــان میدهد که rand-index برابر 97.96% اســـت که نشـــان دهنده خوب بودن کیفیت کلاسترینگ است.

در جهت تعیین لیبل داده های تست، برای اینکه حجم داده های مورد بررسی در الگوریتم KNN را کم کنیم، ابتدا k1 کلاستر نزدیک به آن داده را بدست میاوریم، سپس داده های داخل آن کلاستر ها را به عنوان داده train به الگوریتم KNN نهایی میدهیم.

برای پیدا کردن k1 کلاستر نزدیک به هر داده، مراکز کلاسترها را که قبلا بدست آوردیم (centroids) را به عنوان نماینده آن کلاستر انتخاب میکنیم و سپس فاصله داده را تا مراکز کلاستر ها را با هم مقایسه میکنیم.

تابع find_nearest_clusters_neighbors برای بدست آوردن k1 کلاستر نزدیکتر بر اساس مراکز آنها استفاده میشود.

در این تابع با استفاده از الگوریتم KNN ابتدا یک شئ classifier ایجاد میکنیم و آن را با داده های train فیت میکنیم. x_train در حقیقت حاوی بردار **مراکز** کلاسترهاست. سپس با استفاده از متد ()kneighbors به ازای این داده تست، اندیس نزدیک ترین داده ها (مراکز) به آن را بدست میاوریم. یعنی در حقیقت شماره کلاسترهای نزدیکتر را بدست میاوریم.

```python
def find_nearest_clusters_neighbors(x_train, x_test, k) -> np.ndarray:
    """
    Find k cluster(s) near data `x_test`.

    Parameters:
    k : int
    Number of neighbors to use.

    Returns:
    nearest_indices : ndarray of shape (n_queries, k)
        Indices of the nearest neighbors in the training set.
    """
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(x_train, range(len(x_train)))
    nearest_indices = knn_classifier.kneighbors(x_test, n_neighbors=k, return_distance=False)

    return nearest_indices
```

در تابع gather_clusters_data داده های مربوط به هر کلاستر را از لیست x_train استخراج میکنیم و در key متناظر با شماره کلاستر، در دیکشنری clusters_data ذخیره میکنیم. اینگونه با داشتن شماره کلاستر میتوانیم به داده های داخل آن کلاستر دسترسی داشته.

```
def gather_clusters_data(k_number: int, clusters: np.ndarray) -> Tuple[Dict[int, List], Dict[int, List]]:
    """
    Gather data for each cluster.

    Parameters:
    k_number : int
        Number of clusters.
    clusters : ndarray of shape (n_samples,)
        Cluster labels for each sample in the training set.

    Returns:
    clusters_data : dict
        Dictionary where keys are cluster labels and values are lists of corresponding samples from the training set.
    clusters_data_labels : dict
        Dictionary where keys are cluster labels and values are lists of corresponding labels from the training set.
    """
    clusters_data = {i: [] for i in range(k_number)}
    clusters_data_labels = {i: [] for i in range(k_number)}
    for i, label in enumerate(clusters):
        clusters_data[label].append(x_train[i])
        clusters_data_labels[label].append(y_train[i])
    return clusters_data, clusters_data_labels
```

در تابع find_nearest_clusters_neighbors شماره کلاسترهای نزدیک را پیدا کردیم. همچنین در تابع gather_clusters_data داده های داخل هر کلاستر را به صورت جداگانه جمع آوری کردیم. حال کافیست از دیکشنری خروجی تابع gather_clusters_data، به ازای key های برابر با شماره کلاسترهای نزدیک (nearest_indices)، value های آنها جمع آوری کرده تا به عنوان داده‌ی train به الگوریتم KNN نهایی بدهیم.

این کار را با استفاده از تابع get_nearest_clusters_data انجام میدهیم:

```
def get_nearest_clusters_data(nearest_clusters_indices, clusters_data, clusters_data_labels):
    """
    Access data points in the cluster with centroid index 'nearest_clusters_indices'
    """
    all_data_in_nearest_clusters = []
    all_labels_in_nearest_clusters = []
    for indices in nearest_clusters_indices:
        data_in_nearest_clusters = []
        labels_in_nearest_clusters = []
        for index in indices:
            data_in_nearest_clusters.extend(clusters_data[index])
            labels_in_nearest_clusters.extend(clusters_data_labels[index])
        all_data_in_nearest_clusters.append(data_in_nearest_clusters)
        all_labels_in_nearest_clusters.append(labels_in_nearest_clusters)
    return all_data_in_nearest_clusters, all_labels_in_nearest_clusters
```

این تابع اینگونه عمل میکند که برای هر داده تســـت به ازای اندیس های نزدیک ترین کلاســـترها به آن، مقدار value متناظر با این اندیس ها را از دیکشنری حاوی داده های کلاسترها (clusters_data) بدست می‌آورد و در data_in_nearest_clusters ذخیره میکند. همچنین همین کار را با لیبل های این داده ها نیز میکند. در نهایت داده‌های کلاســـترهای نزدیک به هر داده تســـت بدســـت آمده و به صـــورت جداگانه در all_data_in_nearest_clusters ذخیره میشود.

پس الآن برای هر داده تست، داده های مربوط به نزدیک ترین کلاسترها به آن را داریم.

با استفاده از تابع classify_knn یک classifier فیت میکنیم که داده های train آن همان داده های مربوط به نزدیک ترین کلاسترها به هر داده تست است. اینگونه به ازای هر دیتای تست لیبل پیشبینی شده آن را بر اساس k همسایه نزدیک به آن بدست میاوریم و به لیست y_preds اضافه میکنیم.
Accuracy را نیز بر اساس لیبل های پیش بینی شده و لیبل های اصلی داده ها بدست میاوریم.

```
def classify_knn(x_train, y_train, x_test, y_test, k):
    y_preds = []
    for data_train, label_train, one_data_test in zip(x_train, y_train, x_test):
        knn_classifier = KNeighborsClassifier(n_neighbors=k)
        knn_classifier.fit(data_train, label_train)
        y_preds.append(knn_classifier.predict([one_data_test]))

    accuracy = accuracy_score(y_test, y_preds)
    return y_preds, accuracy
```

در این قسمت کد، این مراحل را به ترتیب به ازای مقادیر مختلف k1(k_clusters) و k2(k_data) انجام میدهیم و نتایج حاصله را برای بررسی بهترین مقادیر آنها بررسی میکنیم:

running classifying steps using different amout of *k_nearest_clusters* and *k_nearest_data*

نتایج را در دیکشنری first_k_results ذخیره میکنیم. برای k1 و k2 بازه مشخص میکنیم:

```
first_k_results = {}
k_clusters = range(5, 15, 1)
k_data = range(3, 8, 1)
```

حال به ازای تمام حالات k1 و k2 لیبل داده های تست را بدست میاوریم. مراحل به صورت زیر است.

به ازای مقادیر مختلف k2، second_k_results را ایجاد میکنیم که نتایج مقادیر مختلف k2 را در آن ذخیره کنیم. اندیس نزدیک ترین کلاستر ها را به ازای همه داده های تست بدست میاوریم:

```
for k_nearest_clusters in k_clusters:
    print("-", k_nearest_clusters, "near cluster")
    second_k_results = {} # first element in each touple is accuracy and the second one is score

    # find nearest clusters neighbors
    x_train_clusters = centroids
    nearest_clusters_indices = find_nearest_clusters_neighbors(x_train_clusters,
                                                               x_test,
                                                               k=k_nearest_clusters)
```

داده های مربوط به کلاستر یکسان را کنار هم جمع آوری میکنیم:

```
# gather clusters data
clusters_data, clusters_data_labels = gather_clusters_data(k_number, clusters)
```

داده های مربوط به نزدیک ترین کلاسترها را برای هر داده تست بدست میاوریم:

```
# get nearest clusters data
all_data_in_nearest_clusters, all_labels_in_nearest_clusters = get_nearest_clusters_data(nearest_clusters_indices,
                                                                                         clusters_data,
                                                                                         clusters_data_labels)
```

حال به ازای مقادیر مختلف k2 داده های تست را کلاسیفای میکنیم و لیبل آنها را بدست میاوریم. با توجه به نتیجه آن، مقدار score را نیز بدست میاوریم و در دیکشنری second_k_results ذخیره میکنیم.
همچنین زمان صرف شده برای کلاسیفای کردن داده های تست با این مقادیر k1 و k2 نیز محاسبه میشود که در تحلیل نتایج از آن استفاده شده است.

```
for k_nearest_data in k_data:
    start_time = datetime.now()
    if k_nearest_data > len(all_data_in_nearest_clusters):
        break

    print("-   -", k_nearest_data)
    # classify test data using knn
    x_train_data = all_data_in_nearest_clusters
    y_train_data = all_labels_in_nearest_clusters
    predictions, accuracy = classify_knn(x_train_data, y_train_data, x_test, y_test, k=k_nearest_data)
    score = (accuracy * 100) + (k_nearest_data * (-0.2))
    # result
    end_time = datetime.now()
    total_time = end_time - start_time
    second_k_results[k_nearest_data] = (accuracy*100, score)
    print(f"Total execution time for k1={k_nearest_clusters} and k2={k_nearest_data} : {total_time} seconds")

first_k_results[k_nearest_clusters] = second_k_results
print(first_k_results[k_nearest_clusters])
```

با استفاده از این تابع پراکندگی داده ها در کلاسترهای مختلف را plot میکنیم:

```
data = x_train
labels = clusters
# Perform t-SNE dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
embedded_data = tsne.fit_transform(data)

colormap = plt.get_cmap('tab10')
markers = ['o', 's', '^', '*', 'D']

plt.figure(figsize=(10, 8))

for i, cluster_label in enumerate(np.unique(labels)):
    cluster_indices = np.where(labels == cluster_label)
    plt.scatter(
        embedded_data[cluster_indices, 0],
        embedded_data[cluster_indices, 1],
        label=f'Cluster {cluster_label}',
        alpha=0.7,
        marker=markers[i % len(markers)],
        color=colormap(i / len(np.unique(labels)))
    )

plt.title('t-SNE Visualization of Clusters')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend()
plt.show()
```

در نهایت مقادیر مختلف score که در first_k_results ذخیره شده است را نمایش میدهیم:

```
import pprint
pprint.pprint(first_k_results)
```

تحلیل نتایج

اولین چیزی باید بدست بیاریم بهترین مقدار برای score است.

$$p(penalty) = -0.2$$
$$score = accuracy + (n * p)$$

همانطور که در کد توضیح داده شد، به ازای مقادیر مختلف k1 در اولین KNN (k1 تعداد **کلاسترهای** نزدیک به داده تست)، و همچنین مقادیر مختلف k2 در دومین KNN (k2 تعداد نزدیک ترین **دیتاهای** داده ی تست)، الگوریتم را اجرا میکنیم و مقدار score را بدست میاوریم. خروجی کد به صورت زیر است:

k1\k2	2	5	8	11	14
3	81.529	83.004	82.038	80.461	79.251
4	81.724	83.713	82.917	81.511	80.521
5	81.919	83.859	83.259	82	80.96
6	81.968	84.128	83.235	82.073	81.058
7	81.846	84.152	83.308	82.244	81.131
8	81.797	84.079	83.357	82.317	81.253
9	81.822	84.054	83.381	82.415	81.351
10	81.871	84.152	83.601	82.537	81.449
11	81.871	84.201	83.65	82.488	81.473
12	81.919	84.25	83.577	82.513	81.522
13	81.895	84.25	83.577	82.464	81.522
14	81.919	84.25	83.625	82.464	81.497

جدول ۱-۱

هر خانه جدول نشان دهنده مقدار score الگوریتم به ازای مقادیر k1 و k2 متناظر با آن است. همچنین در جدول زیر تغییرات score با تغییر شدت رنگ سبز مشخص شده است.

k1\k2	2	5	8	11	14
3	81.529	83.004	82.038	80.461	79.251
4	81.724	83.713	82.917	81.511	80.521
5	81.919	83.859	83.259	82	80.96
6	81.968	84.128	83.235	82.073	81.058
7	81.846	84.152	83.308	82.244	81.131
8	81.797	84.079	83.357	82.317	81.253
9	81.822	84.054	83.381	82.415	81.351
10	81.871	84.152	83.601	82.537	81.449
11	81.871	84.201	83.65	82.488	81.473
12	81.919	84.25	83.577	82.513	81.522
13	81.895	84.25	83.577	82.464	81.522
14	81.919	84.25	83.625	82.464	81.497

جدول ۲-۱

طبق این جدول، با افزایش تعداد کلاستر ها (k1) مقدار score اغلب افزایش میابد.

تغییرات k1 بین k1=7 و k1=12 در حدود **۰.۱** است، پس با توجه به اینکه با انتخاب ۱۲ کلاستر حجم داده‌ی بسیار بیشتری را نسبت به ۷ کلاستر به عنوان داده train برمیگزینیم (نزدیک به ۲ برابر) و این باعث افزایش زمان train میشود و همچنین نتیجه score آنچنان بهبود نمیابد، انتخاب ۷ کلاستر بهترین گزینه است.

همچنین با افزایش تعداد نزدیکترین همسایه ها (k2) از ۲ تا ۵، مقدار score افزایش میابد و از آن به بعد با افزایش آن score کاهش میابد. پس بهترین مقدار برای k2 در بازه ۳ تا ۷ قرار دارد. برای مشخص شدن آن کد را برای این مقادیر اجرا میکنیم. خروجی به صورت جدول زیر است:

k1\k2	3	4	5	6	7
5	83.404	83.693	83.859	83.537	83.142
6	83.526	83.937	84.128	83.732	83.361
7	83.526	83.913	84.152	83.952	83.557
8	83.429	83.937	84.079	83.879	83.63
9	83.478	84.01	84.054	83.903	83.703
10	83.551	84.059	84.152	83.952	83.874
11	83.526	84.108	84.201	83.952	83.899
12	83.526	84.132	84.25	83.952	83.923
13	83.502	84.132	84.25	83.977	83.923
14	83.478	84.132	84.25	84.025	83.923

جدول ۱-۲

در جدول زیر تغییرات score با تغییر شدت رنگ سبز مشخص شده است.

k1\k2	3	4	5	6	7
5	83.404	83.693	83.859	83.537	83.142
6	83.526	83.937	84.128	83.732	83.361
7	83.526	83.913	84.152	83.952	83.557
8	83.429	83.937	84.079	83.879	83.63
9	83.478	84.01	84.054	83.903	83.703
10	83.551	84.059	84.152	83.952	83.874
11	83.526	84.108	84.201	83.952	83.899
12	83.526	84.132	84.25	83.952	83.923
13	83.502	84.132	84.25	83.977	83.923
14	83.478	84.132	84.25	84.025	83.923

جدول ۲-۲

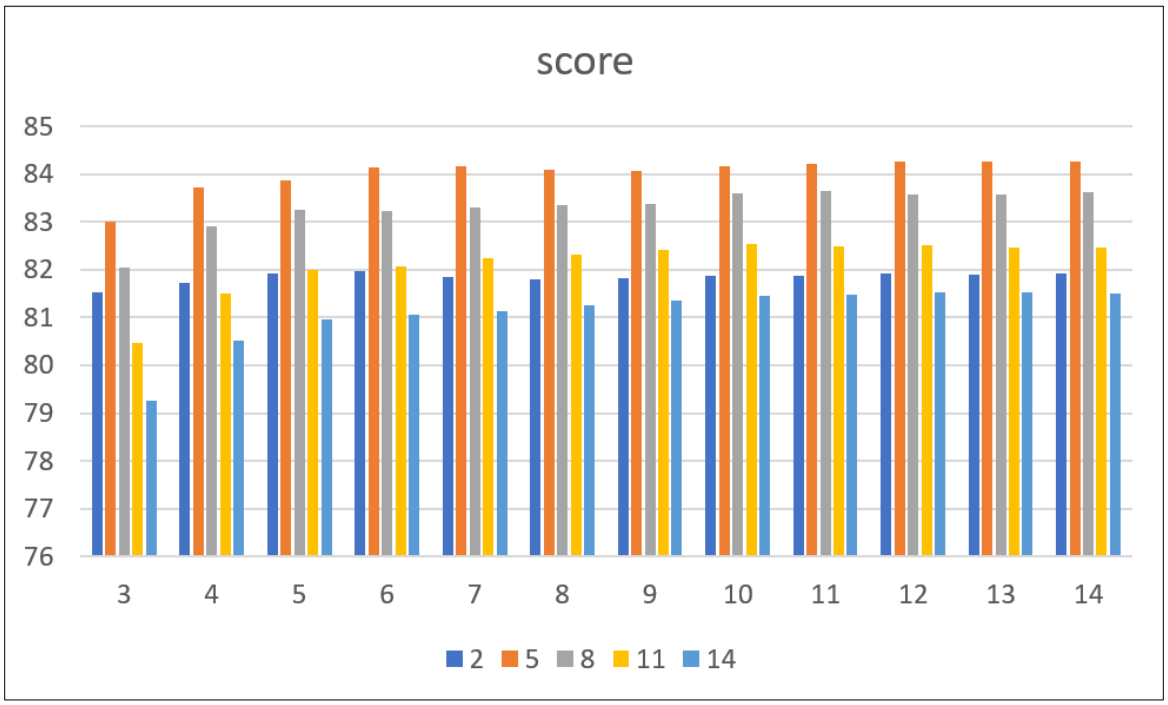
همانطور که در این جدول مشخص شده است، به ازای k1=7 بهترین مقدار برای score در k2=5 اتفاق می‌افتد. منظور از بهترین صرفا دقت بیشتر در پیش‌بینی نیست و بلکه زمان هم در انتخاب ما موثر است. پس بیایید بررسی کنیم به ازای هر k1 و k2 چقدر زمان صرف پیدا کردن لیبل داده تست می‌شود.

k1\k2	3	4	5	6	7
5	0:00:15	0:00:17	0:00:15	0:00:15	0:00:14
6	0:00:19	0:00:19	0:00:21	0:00:19	0:00:18
7	0:00:20	0:00:20	0:00:21	0:00:20	0:00:18
8	0:00:20	0:00:22	0:00:22	0:00:23	0:00:23
9	0:00:26	0:00:25	0:00:25	0:00:25	0:00:25
10	0:00:28	0:00:29	0:00:25	0:00:25	0:00:25
11	0:00:27	0:00:27	0:00:29	0:00:28	0:00:29
12	0:00:32	0:00:32	0:00:31	0:00:33	0:00:32
13	0:00:34	0:00:34	0:00:33	0:00:33	0:00:36
14	0:00:42	0:00:37	0:00:36	0:00:37	0:00:39

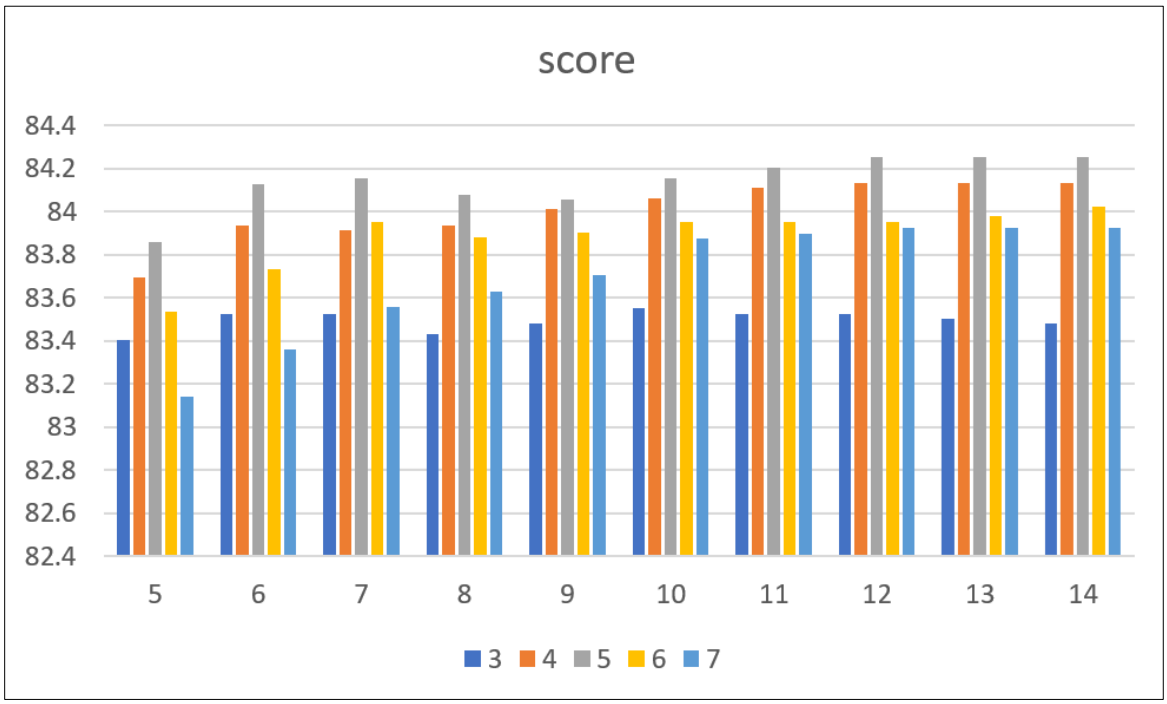
نتیجه میشود که مقادیر k1=7 و k2=5 بهترین نتیجه را براساس مقدار score، حجم داده‌ی train و زمان train بدست میدهد. درست است که با افزایش k1 به بیشتر از ۸ دقت افزایش می‌یابد اما چون به شدت ناچیز است و برای این دقت ناچیز باید ۱۰ ثانیه بیشتر صبر کنیم در نتیجه صرفه زمانی ندارد.

مقایسه score با دو نمودار زیر:

برای جدول ۱-۱



برای جدول ۱-۲



بررسی دلیل بیشتر شدن مقدار score با افزایش مقدار k1

با توجه به اینکه از الگوریتم k-means با k=50 برای کلاستر بندی استفاده میکنیم، چون تعداد لیبل ها (۱۲۰) از تعداد کلاستر ها بیشتر است، پس در هر کلاستر قطعا داده هایی با فیچر مشابه و لیبل متفاوت وجود دارد. در نتیجه طبق اصل لانه کبوتری داده هایی با لیبل یکسان با داده‌ی تست در کلاستر های اطرف وجود دارد که به داده تست نزدیک هستند (به دلیل شباهت) . پس با افزایش تعداد کلاستر های همسایه، این داده ها وارد داده های train میشوند و دقت ما در پیشبینی لیبل تست بالاتر میرود. با این حال افزایش k1 از مقدار معینی به بعد تاثیر چندانی بر روی دقت ندارد زیرا داده های مشابه با داده تست که در کلاستر های اطراف بررسی شده اند در کلاستر های دورتر کمتر پیدا میشوند.

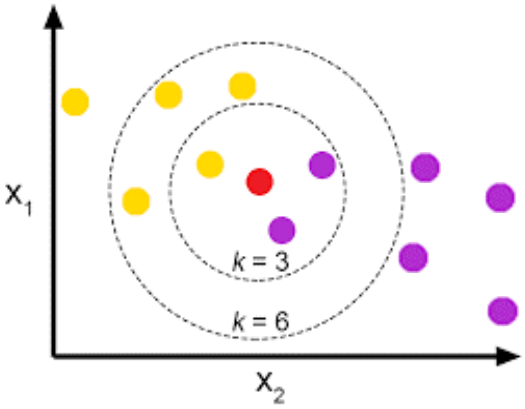
ما k2 داده نزدیک را از داخل k1 کلاستر نزدیک برمیگزینیم و هر کلاستر به طور میانگین ۸۰ داده دارد و مقدار k2 بسیار کوچک تر از آن است، در نتیجه افزایش تعداد کلاستر ها که باعث میشود داده های دورتر جهت بررسی در دامنه ما قرار بگیرند، از جایی به بعد تاثیری روی انتخاب داده های نزدیک و لیبل نهایی نمیگذارد.

بررسی حالتی که با افزایش مقدار k1 مقدار score کاهش میابد

اگر از لیبل داده ای که به عنوان تست استفاده می‌کنیم در داده های train به تعداد کمی وجود داشته باشد، با افزایش k2 در الگوریتم KNN تعداد داده های مشابه ولی با لیبل متفاوت بیشتری در دامنه ما قرار میگیرند. پس

در این صورت اگر k2 کوچکتر باشد، داده هایی که لیبل داده تست را دارند و به اندازه کافی به آن نزدیک هستند، نقش مهمتری را در پیش بینی لیبلش ایفا میکنند (مانند شکل زیر، لیبل درست آن بنفش است).

با توجه به اینکه داده های اولیه‌ی ما از نوع unclustered هستند، برای حل مشکل کمبود داده تست برای لیبلی خاص میتوانیم از data augmentation استفاده کنیم.

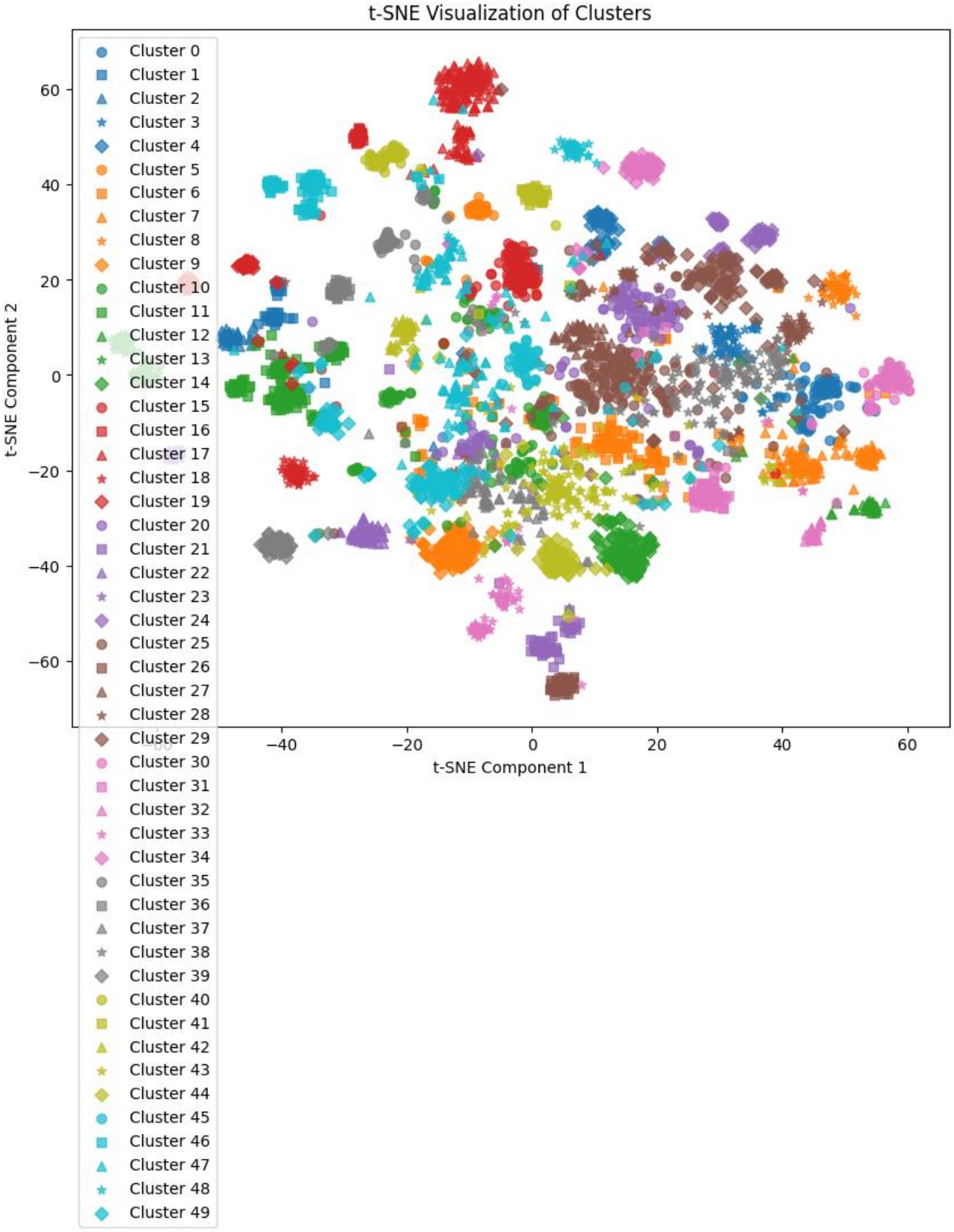


ارتباط بین لیبل های KNN و لیبل های کلاستر

بعد از رسیدن به بهترین امتیاز میخواهیم بررسی کنیم آیا ارتباطی بین لیبل های KNN و لیبل های کلاستر وجود دارد؟ به عبارت دیگر میخواهیم بررسی کنیم هر کلاستر چقدر مقدار در پیش‌بینی الگوریتم KNN کمک کننده بوده و توانسته کلاستری با درصد خلوص بیشتر بدست بیاورد. به این منظور از rand index برای ارزیابی خلوص کلاستر های خود استفاده میکنیم.

خروجی تابع ارزیابی به ما دقت 97.96 درصد را نمایش می‌دهد. این به این معنی است که اغلب داده‌های ما در کلاستر درست و مختص به خود هستند. در نتیجه لیبل داده های داخل کلاسترهای ما تا حدود خوبی بیان کننده یک گل هستند و زمانی که میخواهیم با KNN دادهٔ تستی را پیش‌بینی کنیم، اگر در کنار کلاستر درست قرار بگیرد که در اکثر مواقع همین است میتوانیم به درستی گل را تشخیص بدهیم. مگر آنگه از آن نوع گل دادهٔ زیادی برای تمرین نداده بوده باشیم.

نمایشی از همه کلاسترها را ببینیم:


t-SNE Visualization of Clusters
t-SNE Component 1
t-SNE Component 2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster 10
Cluster 11
Cluster 12
Cluster 13
Cluster 14
Cluster 15
Cluster 16
Cluster 17
Cluster 18
Cluster 19
Cluster 20
Cluster 21
Cluster 22
Cluster 23
Cluster 24
Cluster 25
Cluster 26
Cluster 27
Cluster 28
Cluster 29
Cluster 30
Cluster 31
Cluster 32
Cluster 33
Cluster 34
Cluster 35
Cluster 36
Cluster 37
Cluster 38
Cluster 39
Cluster 40
Cluster 41
Cluster 42
Cluster 43
Cluster 44
Cluster 45
Cluster 46
Cluster 47
Cluster 48
Cluster 49