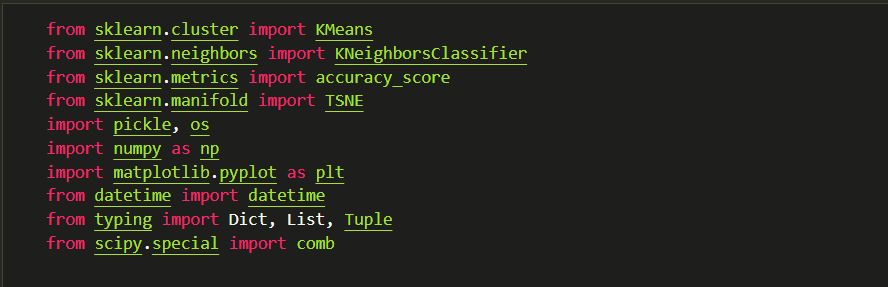
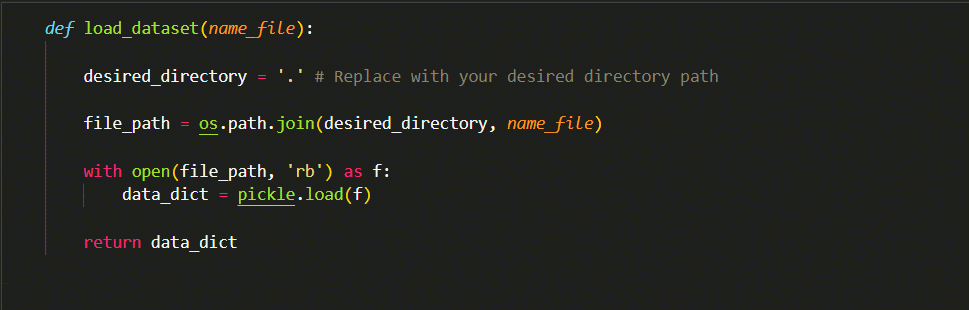
در این فاز از ما خواسته شده که با استفاده از ویژگیهای (features) تهیه شده از تصاویر دیتاست flower102، ابتدا به صورت بدون ناظر (unsupervised) با الگوریتم k-means دیتاها را به صورت مناسب طبقه بندی کرده و سپس در بخش بعد با شکستن داده ها به گروه های کوچکتر و الگوریتم  
 k-Nearest neighbors بهترین امتیاز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

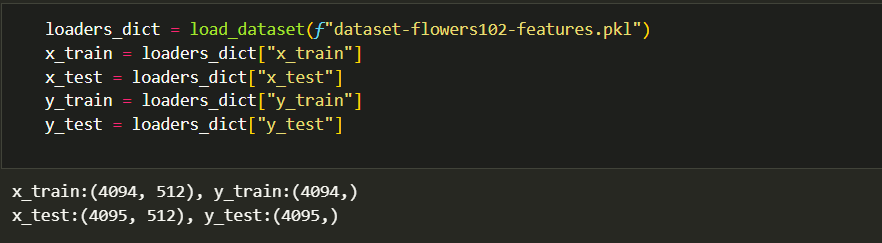
# بررسی کد

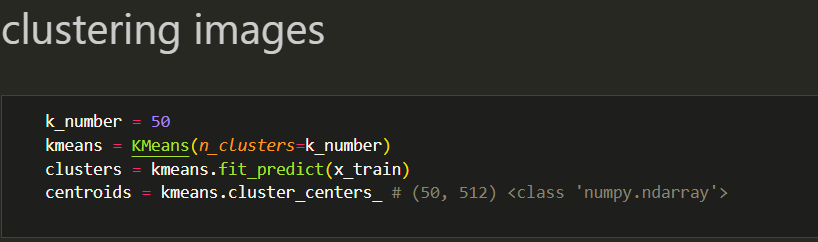
ابتدا کتابخانه های مورد نیاز را ایمپورت میکنیم:

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



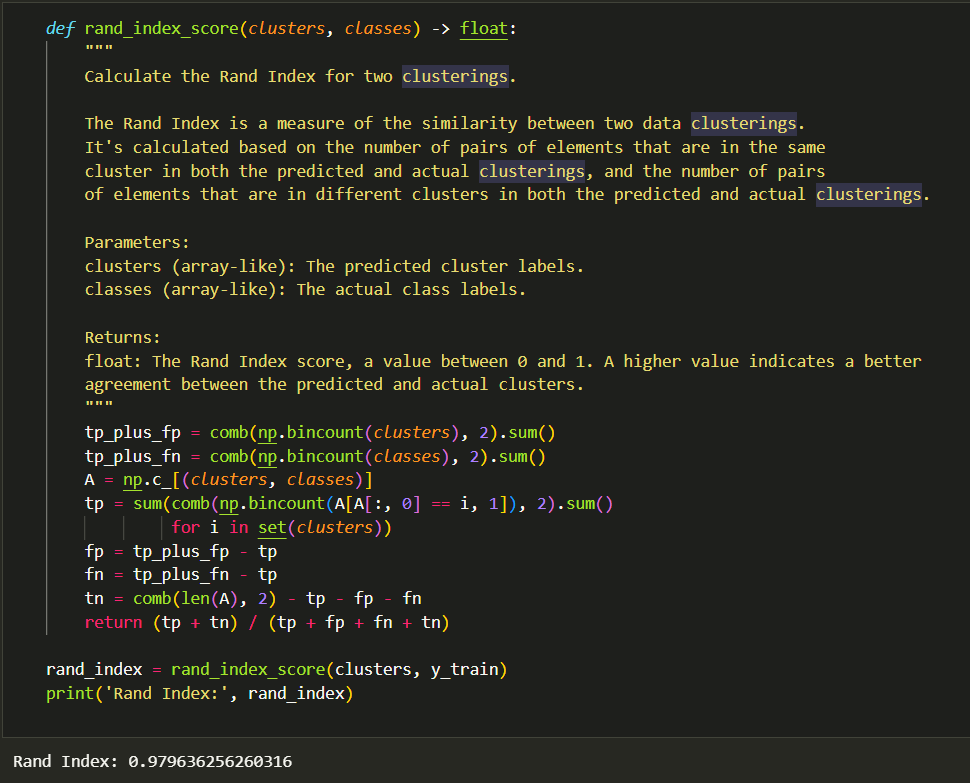
سپس داده های لود شده را به دو بخش test و train تقسیم میکنیم:



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

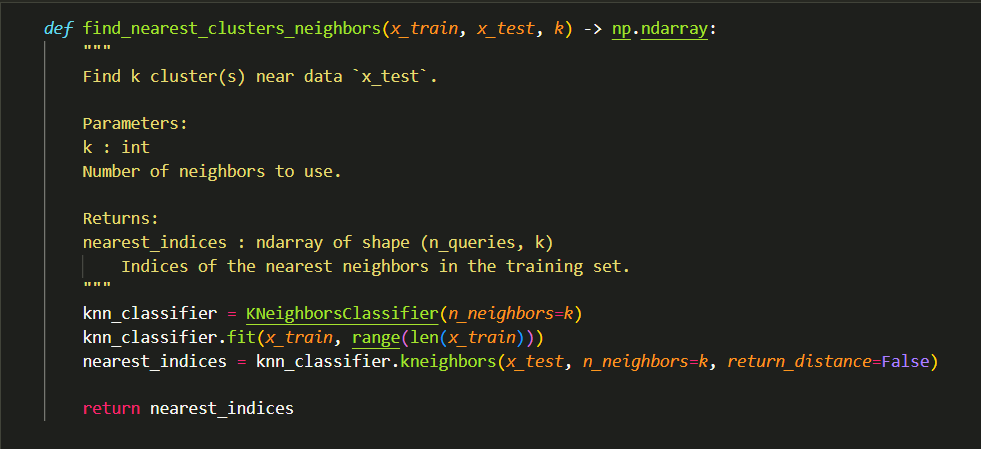
از متریک rand-index برای ارزیابی کیفیت کلاستر بندی الگوریتم k-means استفاده میکنیم. فرمول محاسبه آن به صورت روبرو است که در تابع زیر پیاده سازی شده است:

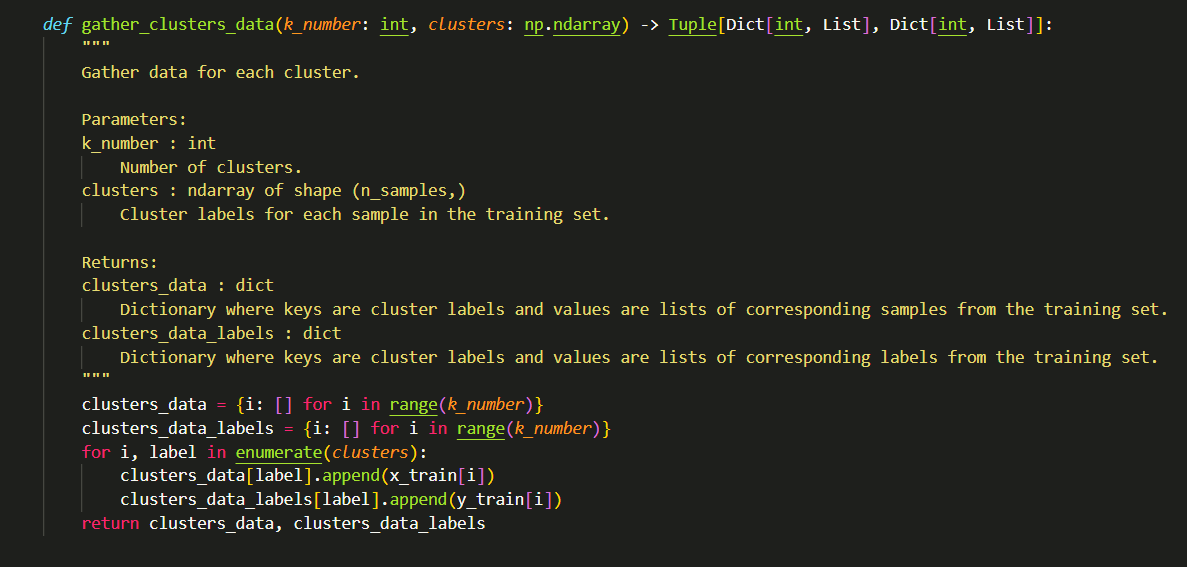
RI= (TP+TN) / (TP+FP+FN+TN)



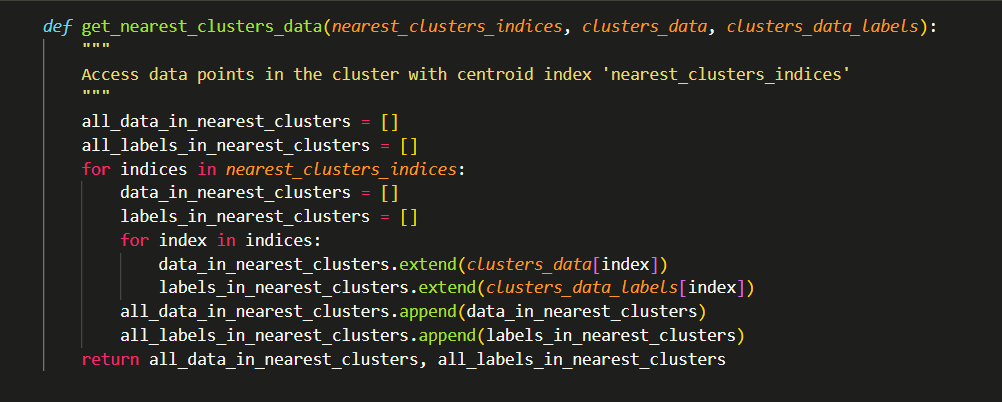
خروجی کد نشان میدهد که rand-index برابر 97.96% است که نشان دهنده خوب بودن کیفیت کلاسترینگ است.

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.



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.

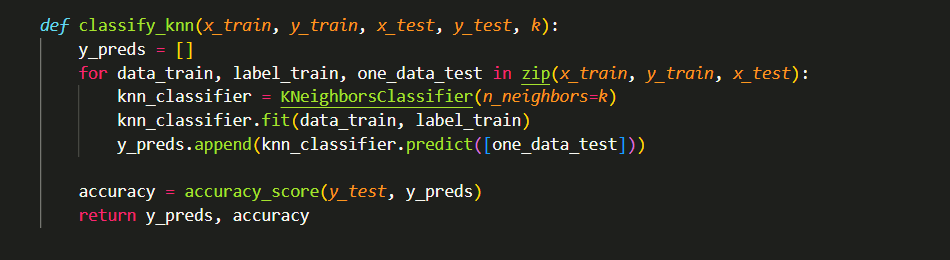
در تابع find\_nearest\_clusters\_neighbors شماره کلاسترهای نزدیک را پیدا کردیم. همچنین در تابع gather\_clusters\_data داده های داخل هر کلاستر را به صورت جداگانه جمع آوری کردیم. حال کافیست از دیکشنری خروجی تابع gather\_clusters\_data، به ازای key های برابر با شماره کلاسترهای نزدیک (nearest\_indices)، 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:



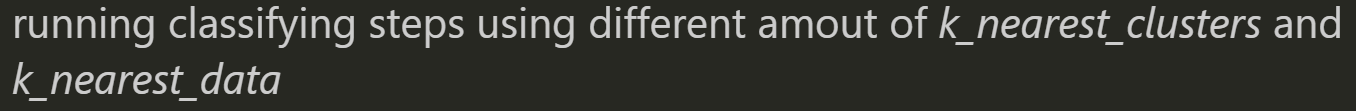
این تابع اینگونه عمل میکند که برای هر داده تست به ازای اندیس های نزدیک ترین کلاسترها به آن، مقدار value متناظر با این اندیس ها را از دیکشنری حاوی داده های کلاسترها (clusters\_data) بدست می‌آورد و در data\_in\_nearest\_clusters ذخیره میکند. همچنین همین کار را با لیبل های این داده ها نیز میکند.  
در نهایت داده‌های کلاسترهای نزدیک به هر داده تست بدست آمده و به صورت 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.



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:

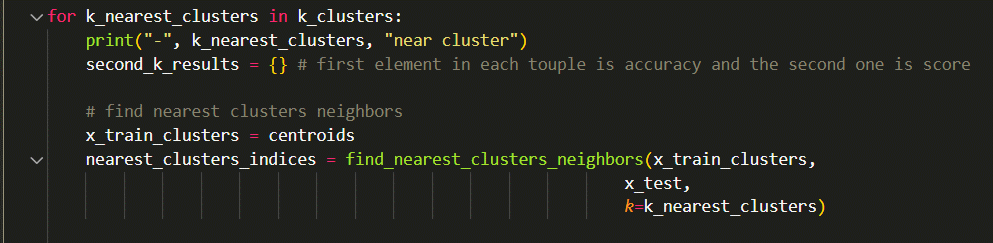


نتایج را در دیکشنری first\_k\_results ذخیره میکنیم. برای k1 و k2 بازه مشخص میکنیم:



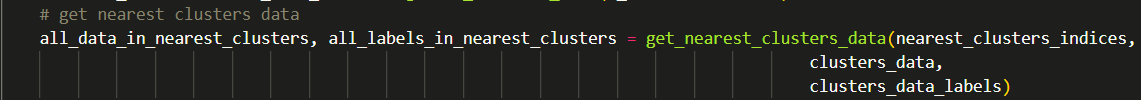
حال به ازای تمام حالات k1 و k2 لیبل داده های تست را بدست میاوریم. مراحل به صورت زیر است.

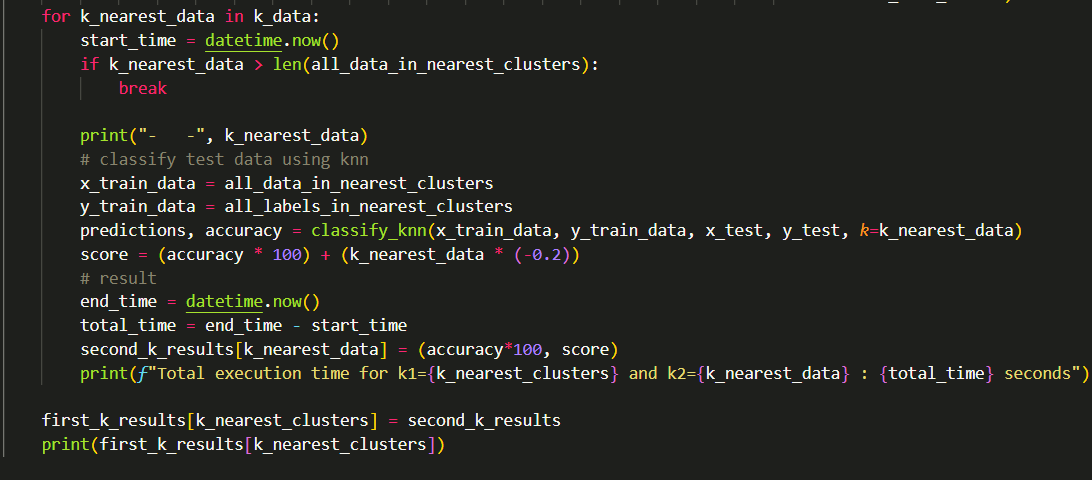
به ازای مقادیر مختلف k2، second\_k\_results را ایجاد میکنیم که نتایج مقادیر مختلف k2 را در آن ذخیره کنیم. اندیس نزدیک ترین کلاستر ها را به ازای همه داده های تست بدست میاوریم:



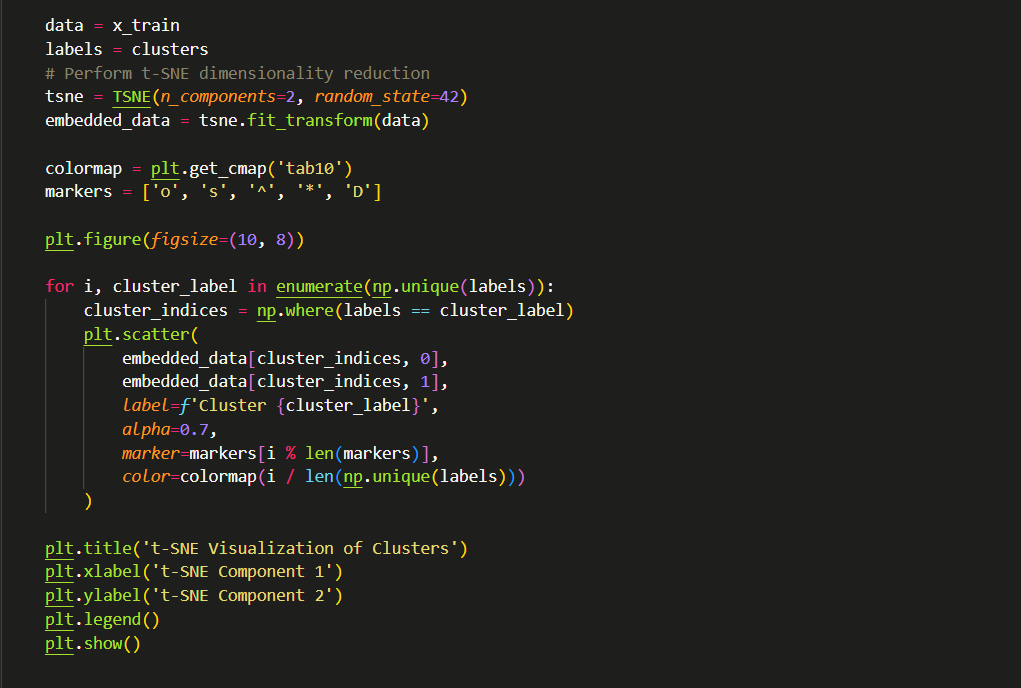
داده های مربوط به کلاستر یکسان را کنار هم جمع آوری میکنیم:

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

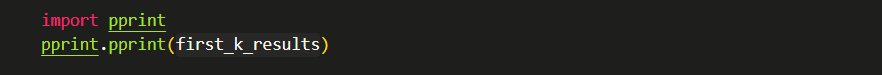


حال به ازای مقادیر مختلف k2 داده های تست را کلاسیفای میکنیم و لیبل آنها را بدست میاوریم. با توجه به نتیجه آن، مقدار score را نیز بدست میاوریم و در دیکشنری second\_k\_results ذخیره میکنیم.  
همچنین زمان صرف شده برای کلاسیفای کردن داده های تست با این مقادیر k1 و k2 نیز محاسبه میشود که در تحلیل نتایج از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

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



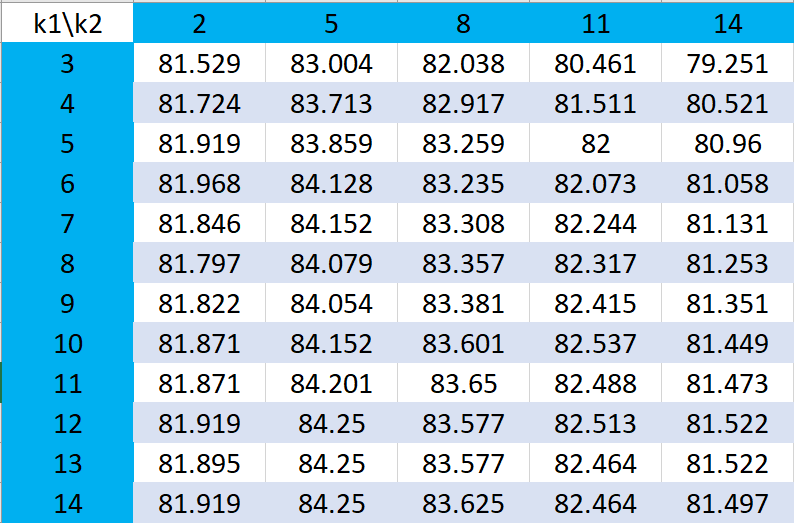
در نهایت مقادیر مختلف score که در first\_k\_results ذخیره شده است را نمایش میدهیم:



# تحلیل نتایج

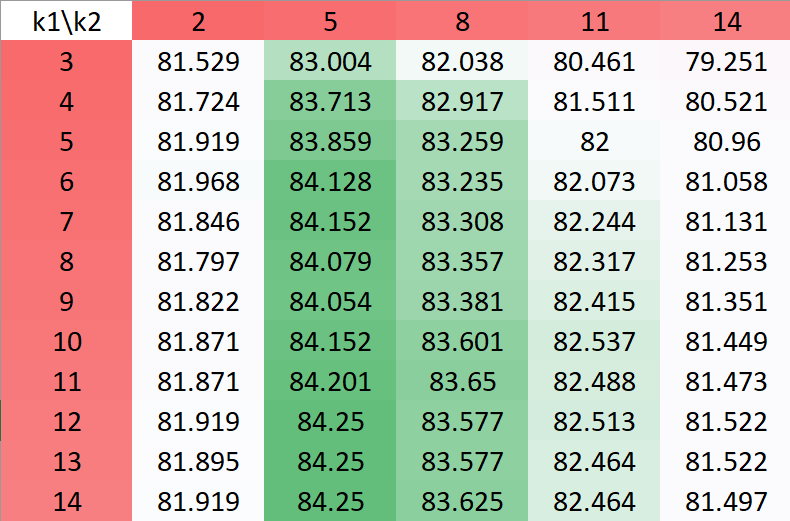
اولین چیزی باید بدست بیاریم بهترین مقدار برای score است.

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:



**جدول 1-1**

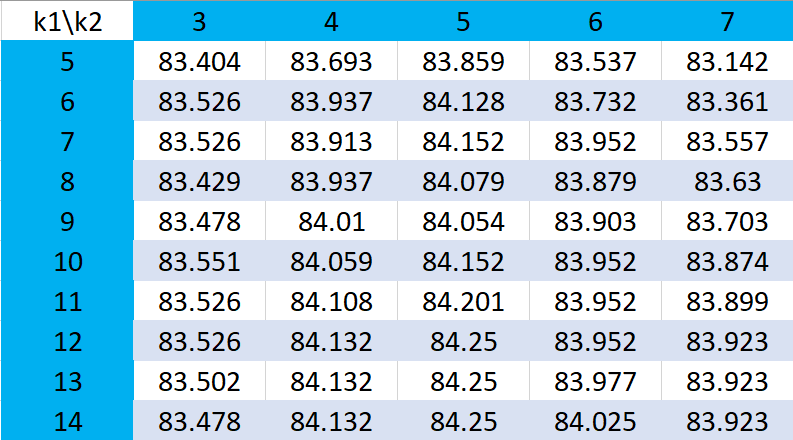
هر خانه جدول نشان دهنده مقدار score الگوریتم به ازای مقادیر k1 و k2 متناظر با آن است. همچنین در جدول زیر تغییرات score با تغییر شدت رنگ سبز مشخص شده است.



**جدول 2-1**

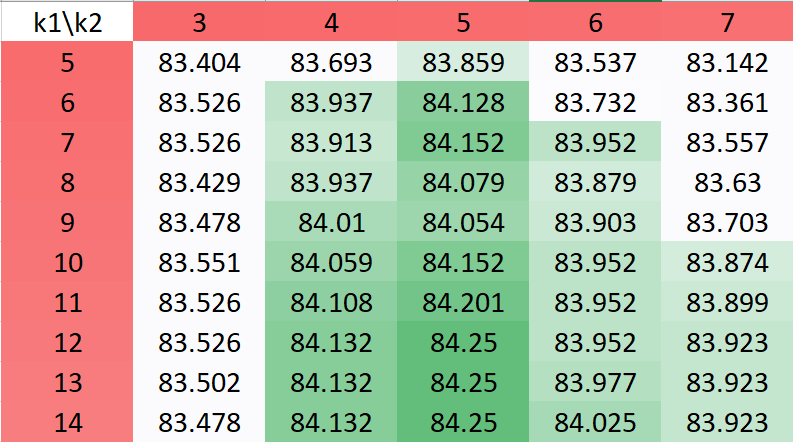
طبق این جدول، با افزایش تعداد کلاستر ها (k1) مقدار score اغلب افزایش میابد.   
تغییرات k1 بین k1=7 و k1=12 در حدود **0.1** است، پس با توجه به اینکه با انتخاب 12 کلاستر حجم داده‌ی بسیار بیشتری را نسبت به 7 کلاستر به عنوان داده train برمیگزینیم (نزدیک به 2 برابر) و این باعث افزایش زمان train میشود و همچنین نتیجه score آنچنان بهبود نمیابد، انتخاب 7 کلاستر بهترین گزینه است.

همچنین با افزایش تعداد نزدیکترین همسایه ها (k2) از 2 تا 5، مقدار score افزایش میابد و از آن به بعد با افزایش آن score کاهش میابد. پس بهترین مقدار برای k2 در بازه 3 تا 7 قرار دارد. برای مشخص شدن آن کد را برای این مقادیر اجرا میکنیم. خروجی به صورت جدول زیر است:



**جدول 1-2**

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



**جدول 2-2**

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.

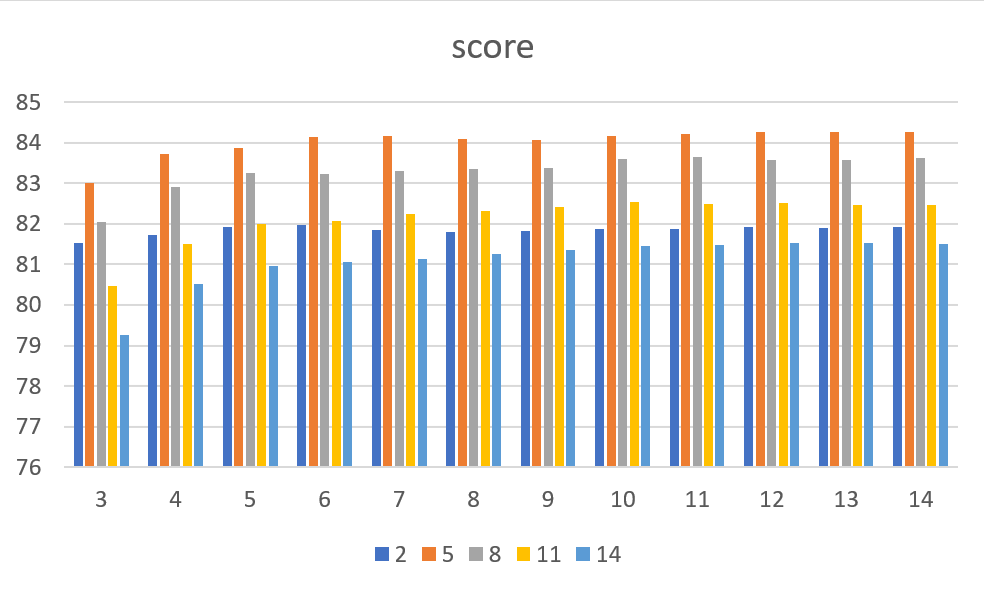
**جدول 3-2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

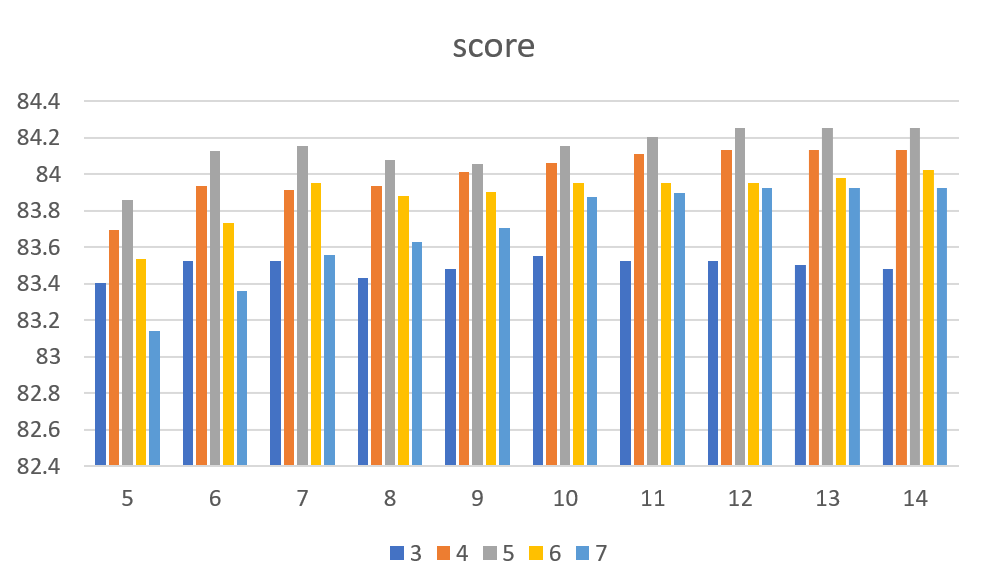
نتیجه میشود که مقادیر k1=7 و k2=5 بهترین نتیجه را براساس مقدار score، حجم داده‌ی train و زمان train بدست میدهد. درست است که با افزایش k1 به بیشتر از 8 دقت افزایش می‌یابد اما چون به شدت ناچیز است و برای این دقت ناچیز باید 10 ثانیه بیشتر صبر کنیم در نتیجه صرفه زمانی ندارد.

مقایسه score با دو نمودار زیر:

برای جدول 1-1



برای جدول 1-2



## بررسی دلیل بیشتر شدن مقدار score با افزایش مقدار k1

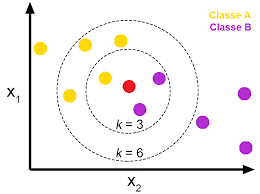
با توجه به اینکه از الگوریتم k-means با k=50برای کلاستر بندی استفاده میکنیم، چون تعداد لیبل ها (120) از تعداد کلاستر ها بیشتر است، پس در هر کلاستر قطعا داده هایی با فیچر مشابه و لیبل متفاوت وجود دارد. در نتیجه طبق اصل لانه کبوتری داده هایی با لیبل یکسان با داده‌ی تست در کلاستر های اطرف وجود دارد که به 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 80 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 دادۀ 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 دادۀ زیادی برای تمرین نداده بوده باشیم.

نمایشی از همه کلاسترها را ببینیم:

