 **بازیابی پیشرفته اطلاعات**

مدرس: دکتر بیگی

شماره گروه: 5

دانشکده مهندسی کامپیوتر تهیه‌کنندگان: نیما جمالی – سپهر فعلی – سینا کاظمی

گزارش فاز دوم پروژه

فهرست مطالب

[ساختن فضای برداری برای دسته‌بندها 3](#_Toc59220303)

[پیش پردازش و تنظیم فیلدهای اولیه 3](#_Toc59220304)

[پیاده‌سازی فضای برداری tf-idf به روش ntn 4](#_Toc59220305)

[ذخیره‌سازی فضای برداری 5](#_Toc59220306)

[پیاده‌سازی دسته‌بندها 6](#_Toc59220307)

[Naïve Bayes 6](#_Toc59220308)

[k-NN 8](#_Toc59220309)

[SVM 10](#_Toc59220310)

[Random Forest 10](#_Toc59220311)

[محاسبه‌ی Validation Set 11](#_Toc59220312)

[محاسبه‌ی بهترین پارامتر برای الگوریتم k-NN 12](#_Toc59220313)

[محاسبه‌ی بهترین پارامتر برای الگوریتم SVM 13](#_Toc59220314)

[بهبود سیستم بازیابی فاز اول پروژه 14](#_Toc59220315)

[تابع اجماع نظر دسته‌بندها برای تعیین برچسب هر مستند 15](#_Toc59220316)

[جستجو بر اساس دسته 16](#_Toc59220317)

[ارزیابی نهایی 17](#_Toc59220318)

[پیاده‌سازی تابع find\_metric 18](#_Toc59220319)

[معیارهای ارزیابی برای Naïve Bayes 18](#_Toc59220320)

[معیارهای ارزیابی برای k-NN 18](#_Toc59220321)

[معیارهای ارزیابی برای SVM 19](#_Toc59220322)

[معیارهای ارزیابی برای Random Forest 19](#_Toc59220323)

[نتیجه‌گیری نهایی معیارهای ارزیابی 20](#_Toc59220324)

[نحوه‌ی تقسیم وظایف 20](#_Toc59220325)

[نیما جمالی 20](#_Toc59220326)

[سپهر فعلی 21](#_Toc59220327)

[سینا کاظمی 21](#_Toc59220328)

[مراجع 21](#_Toc59220329)

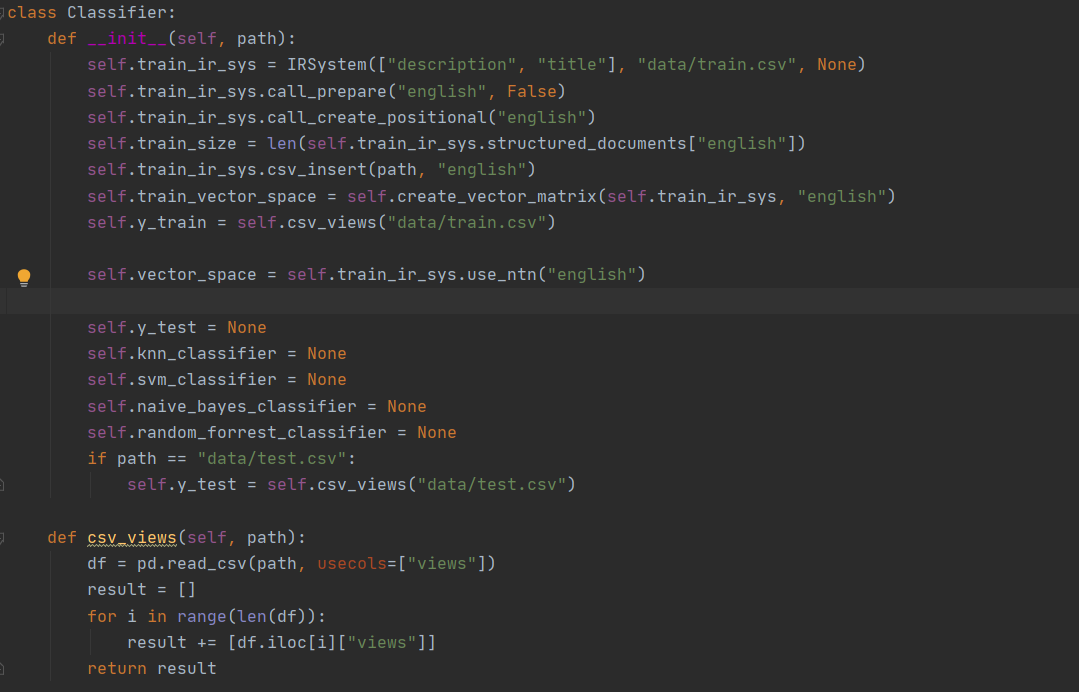
# ساختن فضای برداری برای دسته‌بندها

در این بخش که در واقع بخش آغازین کار است، ابتدا کلاس جدیدی به نام Classifier تعریف می‌کنیم و constructor آن را تکمیل می‌کنیم. همچنین به کلاس IRSystem که در فاز اول پروژه طراحی شده بود، متدی برای محاسبه‌ی فضای برداری اسناد به روش tf-idf اضافه می‌کنیم. در ادامه به شرح پیاده‌سازی این بخش می‌پردازیم.

## پیش پردازش و تنظیم فیلدهای اولیه

کلاس Classifier یک path را به عنوان ورودی دریافت می‌کند. این path مشخص می‌کند که دسته‌بندی قرار است بر روی چه فایلی اعمال شود. سپس از کلاس IRSystem برای پیش‌پردازش مستندات موجود در train.csv استفاده می‌کنیم. تابع call\_prepare و call\_create\_positional را برای این مجموعه مستندات صدا می‌زنیم تا نمایه‌ی positional ساخته شود. از آنجا که می‌خواهیم مستندات موجود در path را هم در این مجموعه درج کنیم، سایز مجموعه‌ی آموزش را در متغیری به نام train\_size نگه می‌داریم. سپس مستندات موجود در path را درج می‌کنیم. y\_test را هم که برای محاسبه‌ی معیارهای ارزیابی استفاده می‌شود، تنها در صورتی تعریف می‌کنیم که path همان test.csv باشد. در این صورت y\_test برابر با لیستی خواهد بود که مقادیر آن، به ترتیب برابر با مقادیر ستون views در داده‌ی آزمون است.

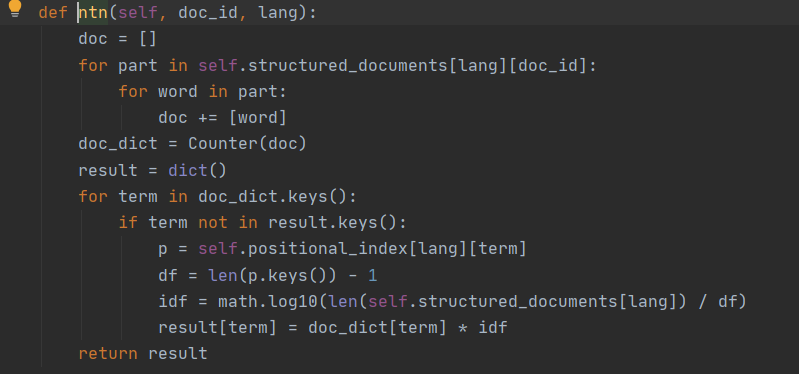
همچنین دو متغیر train\_vector\_space و vector\_space هر دو فضای برداری هستند که مورد اول برای محاسبه‌ی دسته‌بندهای SVM و Random Forest ایجاد شده و مورد دوم برای محاسبه‌ی دسته‌بند k-NN استفاده شده‌است که در ادامه آنها را نیز توضیح می‌دهیم.

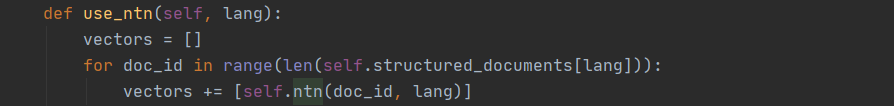


تصویر 1: طراحی اولیه کلاس Classifier و تنظیم فیلدهای آن

## پیاده‌سازی فضای برداری tf-idf به روش ntn

در این بخش، ابتدا در کلاس IRSystem متدی با نام ntn طراحی کردیم که یک doc\_id و زبان آن را ورودی می‌گیرد. سپس یک دیکشنری به نام result می‌سازد که کلیدهای آن، ترم‌های آن مستند و مقادیر آن، حاصل عبارت برای هر کلید است. این تابع توسط تابعی به نام use\_ntn صدا زده می‌شود. این تابع به ازای تمام مستندات، دیکشنری گفته شده را محاسبه کرده و بر می‌گرداند.

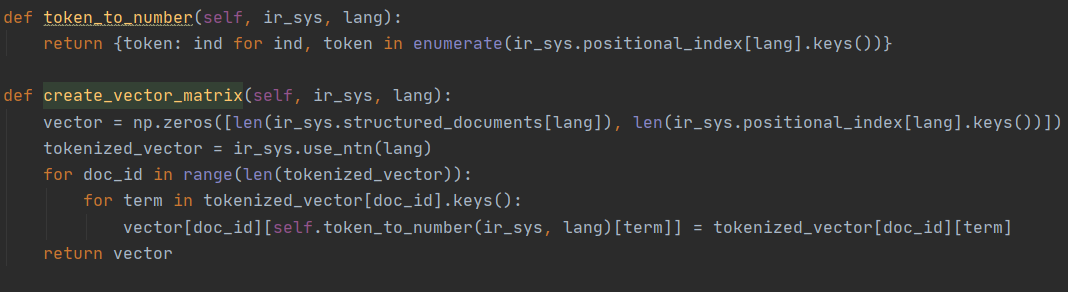




تصویر 2: طراحی توابع ntn و use\_ntn

برای طراحی فضای برداری که الگوریتم‌های کتاب‌خانه‌های آماده برای SVM و Random Forest بتوانند آن را پیاده کنند، نیاز به ماتریسی از ndarrayها داریم و نمی‌توان از دیکشنری استفاده کرد. به همین دلیل نیاز داریم برای هر ترم مستقل در کل دیکشنری (train + test) یک عدد داشته باشیم که به صورت یکتا بین آنها رابطه وجود داشته باشد. برای این کار تابع token\_to\_number را طراحی می‌کنیم که به هر token عدد خاصی نسبت می‌دهد. این عدد خاص از صفر تا len(terms) – 1 می‌تواند باشد.

سپس برای ساختن ماتریسی از ndarrayها، تابع create\_vector\_space را صدا می‌زنیم که ابتدا ماتریسی از صفر تولید می‌کند که تعداد سطرهای آن برابر با تعداد مستندات و تعداد ستون‌های آن برابر با تعداد کل ترم‌هاست. در نهایت با صدا زدن use\_ntn برای هر مستند، برای هر ترم در آن مستند از تابع token\_to\_number استفاده می‌کنیم و وزن مربوط به آن ترم در مستند را در ماتریس به‌روز رسانی می‌کنیم. در نهایت این ماتریس در فیلدی به نام train\_vector\_space ذخیره می‌شود.



تصویر 3: طراحی توابع مورد نیاز برای ساخت فضای برداری به روش ماتریسی

## ذخیره‌سازی فضای برداری

فضای برداری ایجاد شده در قسمت قبل بسیار بزرگ است، زیرا تعداد ترم‌های دیکشنری چیزی در حدود 12000 است! لذا هر نمونه از Classifier که ساخته می‌شود، زمان زیادی می‌گیرد. برای همین تصمیم گرفتیم دو نمونه‌ای که از کلاس Classifier می‌سازیم (یک نمونه برای test.csv و یک نمونه برای ted\_talks.csv) را ذخیره کنیم، البته این ذخیره‌سازی بعد از تکمیل سایر متدهای کلاس Classifier انجام می‌گیرد. با دستور create می‌توانیم یک نمونه ایجاد کنیم و با دستور save، آن نمونه را ذخیره کنیم. نمونه‌ی این دستورات را می‌توان در این [تست](phase2_command) مشاهده کرد.

# پیاده‌سازی دسته‌بندها

در این بخش توابع دسته‌بندی را طراحی می‌کنیم. همچنین توابع مورد نیاز برای محاسبه‌ی بهترین پارامتر را طراحی می‌کنیم. در ادامه به شرح پیاده‌سازی این توابع می‌پردازیم. دقت شود که از کل مجموعه داده‌ی train.csv برای آموزش دسته‌بندها استفاده شده‌است.

## Naïve Bayes

یکی از راه های پیش‌بینی کردن گروه و دسته یک شی(در این پروژه، همان مستند)، الگوریتم Naive-Bayes است. این الگوریتم، تعداد نسبتا زیادی داده تحت عنوان داده‌ی آموزش دریافت می‌کند که گروه هریک از این داده‌ها مشخص است و با پردازش آن، به نوعی ویژگی‌های هر گروه معلوم می‌شود. بدین‌شکل که اگر x رخ داده باشد، احتمال این که به گروه A مرتبط باشد بیشتر است یا گروه B. برای پیاده‌سازی این الگوریتم از سه تابع بهره بردیم:

* تابع naive\_bayes\_train(self, flag\_counter, words, lang): این تابع، قسمت آموزش الگوریتم را انجام می‌دهد و درواقع پایه و اساس کارکرد الگوریتم را شامل می‌شود. اما نحوه کار:

متغیر flag\_counter یک دیکشنری 4عضوی است، متشکل از مقادیر زیر:

* تعداد مستنداتی که عضو گروه 1 هستند.(positive\_docs)
* تعداد مستنداتی که عضو گروه 1- هستند.(negative\_docs)
* تعداد ترم های غیریکتا در مستندات گروه 1 (positive\_terms)
* تعداد ترم های غیریکتا در مستندات گروه 1- (negative\_terms)

متغیر words یک دیکشنری بزرگ از ترم‌های موجود در مجموعه‌ی مستندات داده‌های آموزش است. هر عضو آن، خود یک دیکشنری 2عضوی است که از مقادیر زیر تشکیل شده‌است.

* تعداد تکرار کلمه موردنظر در مستندات گروه 1 (words[term]["positive"])
* تعداد تکرار کلمه موردنظر در مستندات گروه 1- (words[term]["negative"])

این دو متغیر، در این تابع و با پردازش داده‌های آموزش که قبلا پیش‌پردازش شده‌اند و در متغیری از کلاس ir\_system به نام structured\_documents[lang] ذخیره شده‌اند، مقداردهی و تکمیل می‌شوند تا در ادامه مورد استفاده قرار گیرند.

* تابع naive\_bayes\_test(self, flag\_counter, words, lang): این تابع، قسمت آزمایش الگوریتم را انجام می‌دهد و درواقع، خروجی الگوریتم است. یعنی با استفاده از نتایج برآمده از پردازش داده‌های آموزش در تابع naive\_bayes\_train، گروه‌های داده‌های آزمایش را پیش‌بینی می‌کند. اما نحوه‌ی کار چگونه است:

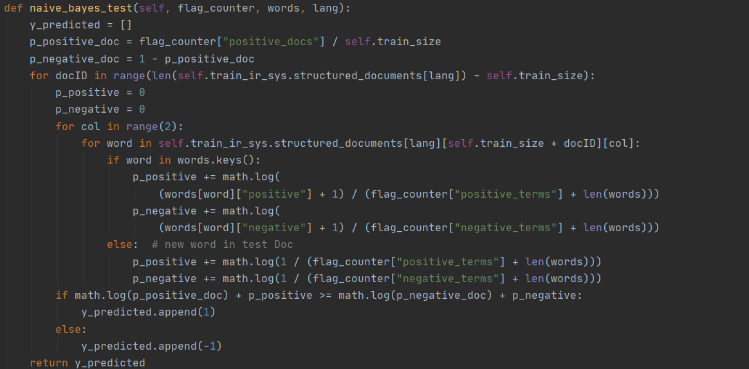
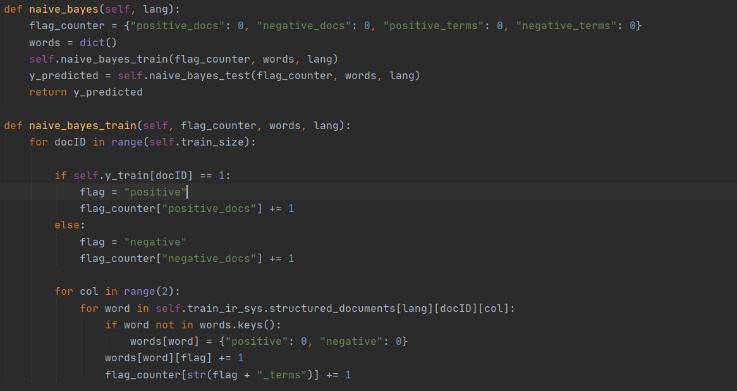
به ازای هر مستند ورودی، به روش زیر می‌توانیم گروه مربوط به آن را حدس بزنیم.

پیاده‌سازی این روابط در کد به این صورت است که

* برای محاسبه به words[t][c] رجوع می‌کنیم.
* برای محاسبه به flag\_counter[c\_terms] رجوع می‌کنیم.
* مقدار B برابر با تعداد ترم‌های یکتای داده‌های آزمون است.
* برای محاسبه نیز به flag\_counter[c\_docs] مراجعه می‌کنیم.

در نهایت، نتیجه حدس‌های خود را در لیستی به سایز داده‌های آزمایش به نام y\_predicted می‌ریزیم که این متغیر، خروجی تابع خواهد بود.

* تابع naive\_bayes(self, lang): این تابع فرآیند اجرای الگوریتم را کنترل می‌کند. یعنی دیکشنری‌های flag\_counter و words را initialize می‌کند، سپس ابتدا تابع آموزش یعنی naive\_bayes\_train و سپس تابع آزمایش یعنی naive\_bayes\_test را صدا می‌کند و خروجی، یعنی y\_predicted را برمی‌گرداند.



تصویر 4: توابع مورد نیاز برای دسته‌بند Naïve Bayes

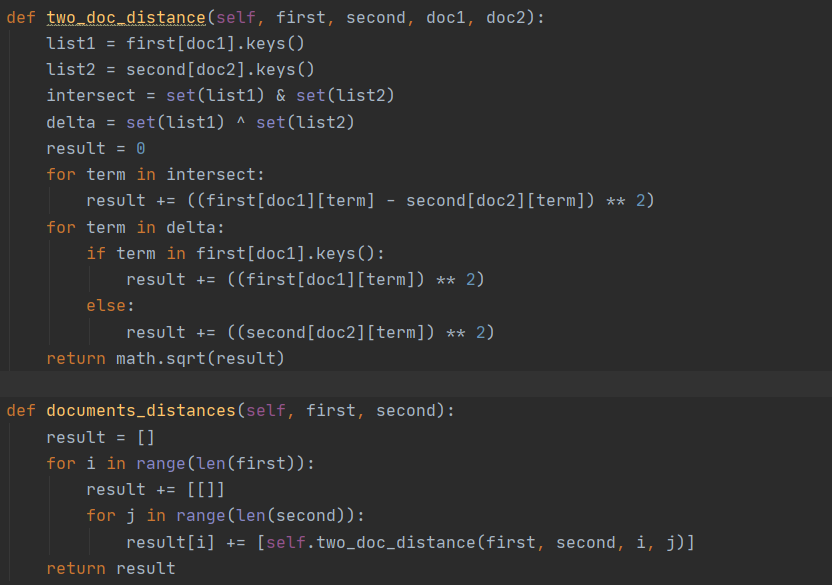
در کنسول می‌توان با دستور naive\_bayes test، برچسب‌هایی که این دسته‌بند برای مجموعه‌ی تست می‌زند مشاهده کرد.

## k-NN

دسته‌بند k-NN بسته به پارامتر k ورودی، k نزدیک‌ترین همسایه‌ی مستند داده شده را در مجموعه‌ی آموزش می‌یابد و با توجه به اکثریت برچسب‌ها در میان آن k مستند، برچسب مستند آزمایشی را محاسبه می‌کند.

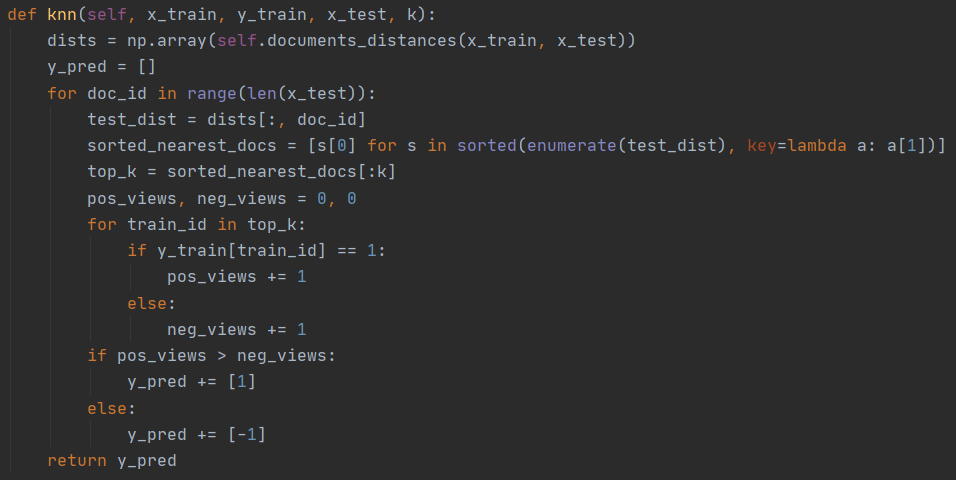
به این منظور تابعی به نام knn طراحی می‌کنیم که مجموعه‌ی مستندات آموزش و برچسب‌های آنها، مجموعه‌ی مستندات آزمایش و پارامتر k را ورودی می‌گیرد. می‌توانستیم از فضای برداری که در بخش [پیاده‌سازی فضای برداری tf-idf به روش ntn](#_پیاده‌سازی_فضای_برداری) توضیح داده شد، استفاده کنیم، اما دسته‌بندی با آن روش به علت این که صفرهای زیادی را بدون آن که در محاسبه نیاز شوند، نگه می‌داریم و در واقع ماتریس sparse است، زمان زیادی می‌گیرد.

به همین علت فضای برداری جدیدی با نام vector\_space (یا به عبارت بهتر knn\_vector\_space) ایجاد می‌کنیم که برای محاسبه‌ی آن، تابع use\_ntn را صدا می‌زنیم. سپس باید فاصله‌ی دو مستند را حساب کنیم. برای این کار تابعی به نام two\_doc\_distance تعریف می‌کنیم که دو مستند را از فضای برداری vector\_space می‌گیرد و فاصله‌ی آن‌ها را محاسبه می‌کند. برای محاسبه‌ی k نزدیک‌ترین همسایه، باید فاصله‌ی هر مستند آزمایش با هر مستند آموزشی را داشته باشیم که این کار از طریق تابع documents\_distances انجام می‌شود و خروجی آن ماتریسی است که سطرهای آن، مستندات آموزش و ستون‌های آن، مستندات آزمایش است.



تصویر 5: توابع محاسبه‌کننده‌ی فاصله‌ی مستندات

در تابع knn، ابتدا این ماتریس محاسبه می‌شود. سپس هر ستون را که متناظر یک مجموعه‌ی تست است، به صورت صعودی مرتب کرده و k تای اول را برای هر ستون انتخاب می‌کنیم. سپس برچسب هر یک از این k مستند را بررسی می‌کنیم. اگر تعداد 1 ها بیشتر باشد، برچسب مستند آزمایش هم 1 و در غیر این صورت 1- خواهد بود.



تصویر 6: تابع محاسبه‌ی دسته‌بند k-NN

در کنسول هم می‌توان با دستور knn test k برچسب‌هایی که دسته‌بند k-NN به داده‌های تست می‌زند، مشاهده کرد. دقت شود که k یک عدد طبیعی است که کاربر ورودی می‌دهد.

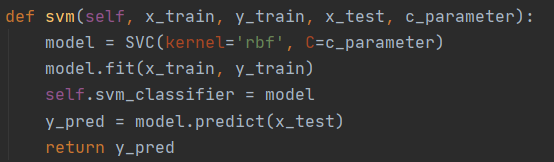
## SVM

بردار های پشتیبان به زبان ساده، مجموعه‌ای از نقاط در فضای n بعدی داده‌ها هستند که مرز دسته‌ها را مشخص می‌کنند و دسته‌بندی داده‌ها بر اساس آنها انجام می‌شود و با جابه‌جایی یکی از آنها خروجی دسته بندی ممکن است تغییر کند.

در این قسمت از کتابخانه‌ی یادگیری ماشین پایتون به نام sklearn استفاده کردیم که تمام کرنل‌ها و توابع نگاشت را به صورت آماده دارد. ۳ تابع LinearSVC , NuSVC , SVC وظیفه اصلی دسته‌بندی را در این کتابخانه دارند که ما از SVC که مخفف Support Vector Classifier است استفاده می‌کنیم.

ورودی‌های این تابع کرنل مورد استفاده و پارامتر C است. کرنل را [گاوسی](https://en.wikipedia.org/wiki/Radial_basis_function_kernel) (rbf) در نظر گرفتیم و پارامتر C را از ورودی می‌گیریم. با استفاده از کرنل گاوسی، داده‌ها را به فضای بی‌نهایت بعدی نگاشت می‌کنیم و در آن فضا دسته‌بندی را انجام می‌دهیم. پارامتر C نیز در واقع میزان اهمیت به اشتباه دسته‌بندی شدن داده‌ها و فاصله‌ی خط جداساز از Support Vectorها را مشخص می‌کند.

در کنسول می‌توان با وارد کردن دستور svm test c که c پارامتری است که کاربر ورودی می‌دهد، برچسب‌های خروجی دسته‌بند SVM را برای مجموعه‌ی تست مشاهده کرد.



تصویر 7: دسته‌بند SVM

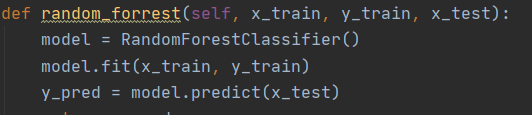
## Random Forest

الگوریتم جنگل تصادفی یا همان Random Forest یک الگوریتم ترکیبی (ensemble) است که از درخت‌های تصمیم بهره می‌گیرد.

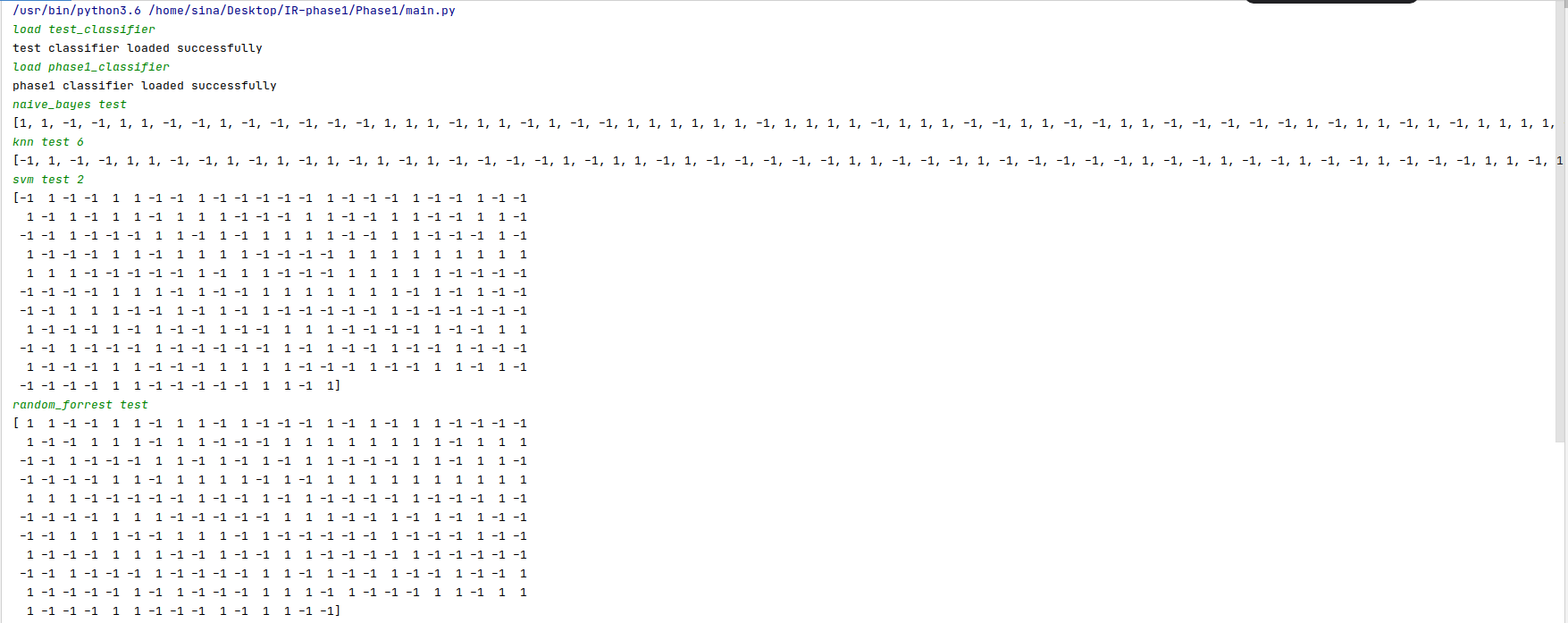
الگوریتم درخت تصمیم (decision trees) می‌تواند به راحتی عملیات طبقه‌بندی را بر روی داده‌ها انجام دهد. حال در الگوریتم جنگل تصادفی از چندین درخت تصمیم (برای مثال ۱۰۰ درخت تصمیم) استفاده می‌شود. در واقع مجموعه‌ای از درخت‌های تصمیم با هم یک جنگل را تولید می‌کنند و این جنگل می‌تواند تصمیم‌های بهتری را نسبت به یک درخت اتخاذ نماید.

در الگوریتم جنگل تصادفی به هر کدام از درخت‌ها، زیرمجموعه‌ای از داده‌ها تزریق می‌شود. برای مثال اگر مجموعه داده دارای ۲۵۵۰ سطر باشد و ۱۰ ستون داشته باشد، الگوریتم جنگل تصادفی به هر کدام از درخت ها ۱۰۰ سطر و ۵ ستون -که به طور تصادفی انتخاب شده‌اند و زیرمجموعه‌ای از مجموعه‌ی داده‌ها هست- می‌دهد. این درخت‌ها با همین دیتاست زیرمجموعه، می‌توانند تصمیم بگیرند و مدل دسته‌بند خود را بسازند.

برای پیاده‌سازی این بخش، از کتابخانه‌ی sklearn استفاده می‌کنیم و از کلاس ensemble که مربوط به الگوریتم‌های ترکیبی است، تابع RandomForestClassifier را برای دسته‌بندی مورد استفاده قرار می‌دهیم و با استفاده از تابع fit فاز آموزش را بر روی داده‌های آموزش انجام می‌دهیم و سپس با استفاده از predict، برچسب داده‌های آزمایش را معین می‌کنیم. با استفاده از دستور random\_forrest test می‌توان برچسب‌ها را در کنسول مشاهده کرد.



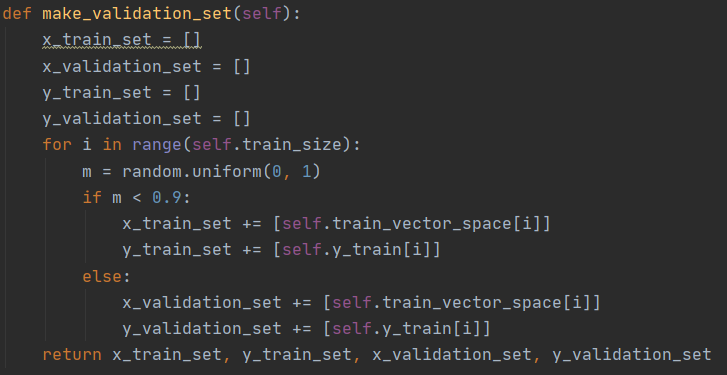
تصویر 8: تابع دسته‌بند random\_forrest



تصویر 9: برچسب‌گذاری هر یک از دسته‌بندها بر روی مجموعه داده‌ی آزمایش

## محاسبه‌ی Validation Set

تابع make\_validation\_set و make\_knn\_validation\_set برای این قسمت پیاده‌سازی شده‌است. هر دوی این توابع عملکرد یکسانی دارند و به ازای هر مستند در فضای برداری یک عدد رندوم با توزیع یکنواخت تولید می‌کنیم تا با احتمال 0.1 مستند در مجموعه‌ی Validation و با احتمال 0.9 در مجموعه‌ی آموزش قرار بگیرد. سپس دو مجموعه را باز می‌گردانیم.

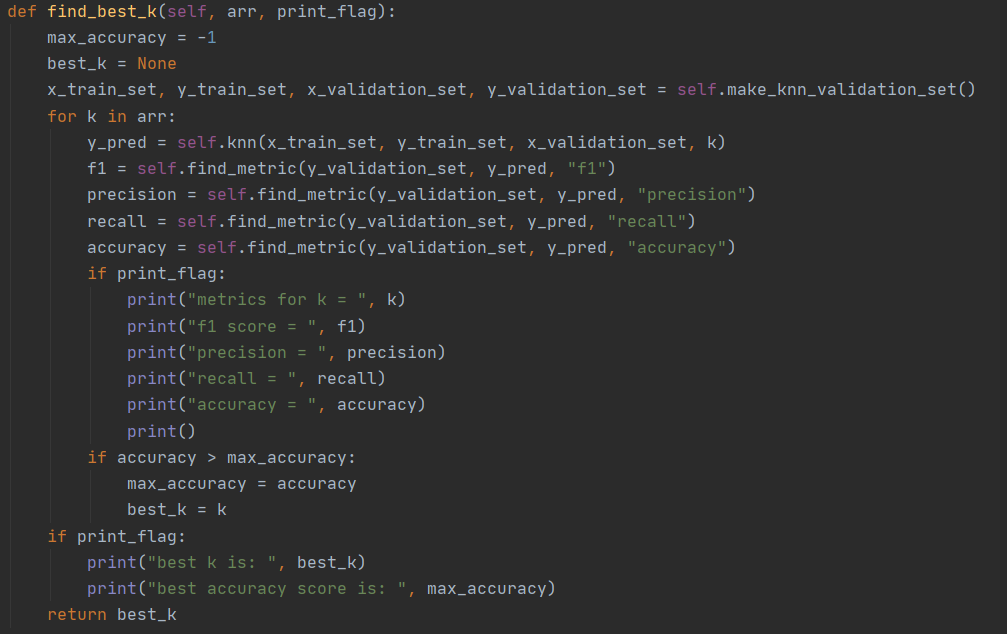


تصویر 10: تابع ایجاد مجموعه‌ی Validation از مجموعه‌ی آموزش

## محاسبه‌ی بهترین پارامتر برای الگوریتم k-NN

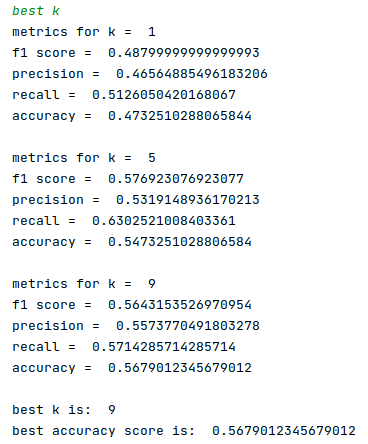
در این بخش از میان kهای گفته شده در صورت پروژه –یعنی 1 و 5 و 9- بهترین k را انتخاب می‌کنیم و در قسمت‌های بعدی نیز از آن استفاده می‌کنیم.

برای این کار از تابع find\_best\_k استفاده می‌کنیم. این تابع در صورتی که فلگ print برابر با True باشد، به ازای هر k از مقادیر گفته شده، تابع knn را با توجه به مجموعه‌ی آموزش و Validation صدا می‌زند و سپس با استفاده از خروجی آن و تابع find\_metric که در بخش [پیاده‌سازی تابع find\_metric](#_پیاده‌سازی_تابع_find_metric) توضیح داده شده، معیارهای مختلف را به ازای هر k در خروجی چاپ می‌کند. در نهایت بهترین k را با توجه به بیشترین مقدار accuracy انتخاب کرده و چاپ می‌کنیم. در صورتی هم که فلگ print برابر False باشد، تنها بهترین مقدار k محاسبه می‌شود و چیزی چاپ نمی‌شود.



تصویر 11: پیاده‌سازی تابع یافتن بهترین پارامتر k

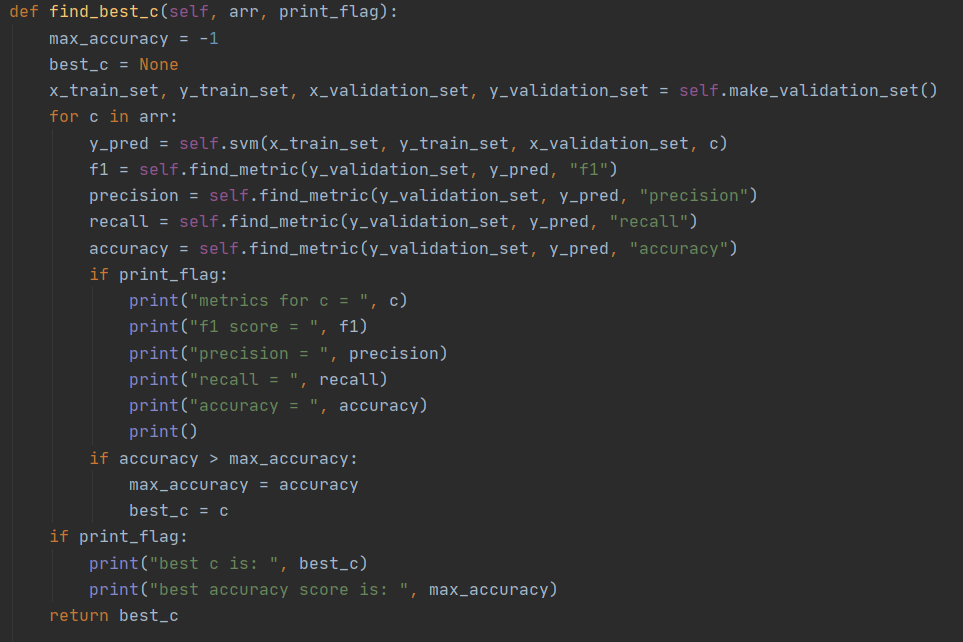
در کنسول می‌توان با دستور best k خروجی این تابع را مشاهده کرد. مشاهده می‌شود که k = 9 بهترین k بوده‌است.



تصویر 12: خروجی دستور best k

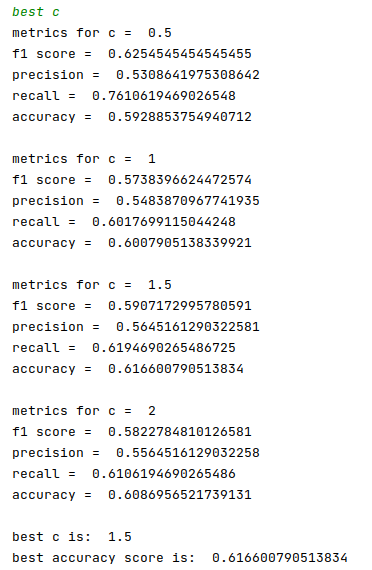
## محاسبه‌ی بهترین پارامتر برای الگوریتم SVM

برای پیاده‌سازی این بخش از تابع find\_best\_c استفاده شده که عینا مشابه تابع find\_best\_k عمل می‌کند که در بخش [قبل](#_محاسبه‌ی_بهترین_پارامتر) توضیح داده شد. با این تفاوت که به جای تابع knn، تابع svm صدا زده می‌شود و مقدار بهترین C از بین مقادیر 0.5، 1، 1.5 و 2 انتخاب می‌شود.



تصویر 13: تابع یافتن بهترین پارامتر C

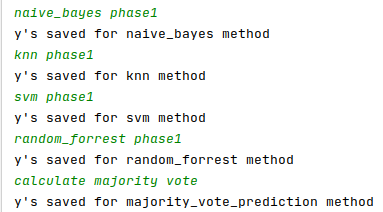
با دستور best c می‌توان خروجی مورد نظر را مشاهده کرد که نشان می‌دهد C = 1.5 بهترین مقدار بوده‌است.



تصویر 14: خروجی دستور best c

# بهبود سیستم بازیابی فاز اول پروژه

در این قسمت، ابتدا با استفاده از داده‌های آموزشی هر چهار دسته‌بند را آموزش می‌دهیم. سپس داده‌های فاز ۱ را که در phase1\_classifier موجودند، با استفاده از هر چهار دسته‌بند برچسب می‌زنیم و ذخیره می‌کنیم. این ذخیره‌سازی به کمک دستورات تصویر زیر انجام می‌شود.



تصویر 15: دستورات لازم برای اجرای دسته‌بندها بر روی مستندات فاز اول

دستور calculate majority vote رای‌های مستندات را مجتمع می‌کند و بهترین برچسب را انتخاب می‌کند. نحوه‌ی کار آن در بخش [تابع اجماع نظر دسته‌بندها برای تعیین برچسب هر مستند](#_تابع_اجماع_نظر) توضیح داده می‌شود. سپس این برچسب‌ها را در فایلی ذخیره می‌کند.

حال می‌توانیم با توجه به ورودی کاربر که یا 1 یا 1- است، مستندات مرتبط با آن دسته را در خروجی به کاربر نمایش دهیم. به این صورت که پس از فیلتر کردن مستندات فاز اول بر اساس دسته، مطابق همان چیزی که در فاز اول پیاده‌سازی شده‌است، مستندات را بر اساس معیار ltc-lnc مرتب‌سازی می‌کنیم و به کاربر نمایش می‌دهیم. در ادامه توابع این بخش را توضیح می‌دهیم.



تصویر 16: فایل‌های جانبی ذخیره شده که برچسب‌های هر دسته‌بندی را نگه می‌دارند.

## تابع اجماع نظر دسته‌بندها برای تعیین برچسب هر مستند

در این قسمت، ابتدا برچسب‌های ذخیره‌سازی شده برای هر دسته‌بند را بارگذاری می‌کنیم و سپس برای هر مستند نظر اجماع دسته‌بندها را به عنوان برچسب آن مستند در نظر می‌گیریم.

نکته‌ای که باید به آن توجه کرد این است که چهار دسته‌بند داریم و لذا ممکن است دو دسته‌بند نظر به برچسب ۱ داشته باشند و دو تای دیگر نظر به برچسب 1-. در این حالت برچسبی که دسته‌بند Random Forest برای مستند انتخاب کرده‌است به عنوان نتیجه‌ی نهایی در نظر گرفته می‌شود، زیرا با آن به دقت بالاتری می‌رسیدیم. همان‌طور که گفته شد، این اجماع نظر به کمک دستور calculate majority vote انجام می‌گیرد.

## جستجو بر اساس دسته

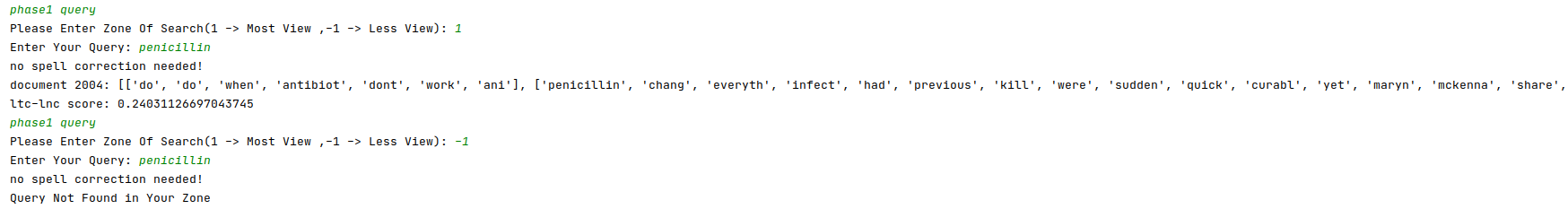
تابع این بخش، phase1\_query است که در IRSystem طراحی شده‌است. ابتدا نظر جمعی دسته‌بندها را لود می‌کنیم و از آن برای برچسب زدن هر مستند فاز اول استفاده می‌کنیم. سپس مانند فاز قبلی پروژه عمل می‌کنیم، با این تفاوت که جستجو را در دسته‌ی مشخص شده انجام می‌دهیم.

در این قسمت برای جستجو با انتخاب دسته‌ی مورد نظر دستور phase1 query در کنسول زده می‌شود. سپس از کاربر می‌خواهیم دسته‌ی مورد نظر را انتخاب کند و بعد از آن مستندات را با توجه به ورودی کاربر فیلتر می‌کنیم. ‌پس از آن کاربر می‌بایست پرسمان خود را وارد کند و پس از تصحیح پرسمان و رتبه‌بندی مطابق فاز قبل، نتایج به کاربر نشان داده می‌شود.

نمونه‌هایی از اجرای این بخش در ادامه دیده می‌شود.



تصویر 17: اجرای دستور phase1 query برای هر دو دسته با پرسمان dianna cohen



تصویر 18: اجرای دستور phase1 query برای هر دو دسته با پرسمان penicillin

# ارزیابی نهایی

در این بخش ابتدا تابع find\_metric را پیاده‌سازی می‌کنیم که با توجه به رشته‌ی ورودی که همان معیار مورد نظر است، مقدار آن معیار را با توجه به دو ورودی دیگر که دو لیست از مقادیر واقعی و مقادیر پیش‌بینی شده هستند، خروجی می‌دهد. سپس برای هر کدام از دسته‌بندها معیارهای خواسته شده را محاسبه می‌کنیم که در ادامه توضیح داده می‌شوند.

## پیاده‌سازی تابع find\_metric

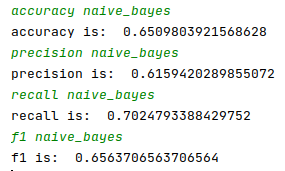
تابع find\_metric سه ورودی به نام‌های y\_test، y\_pred و metric دارد که:

* ورودی y\_test شامل گروه‌های واقعی مستندات آزمایش (test.csv) است.
* ورودی y\_pred شامل گروه‌های پیش‌بینی شده‌ی مستندات آزمایش است.
* ورودی metric نیز معیار ارزیابی را مشخص می‌کند که یک رشته است و مقدار آن برابر با یکی از مقادیر f1، accuracy، precision یا recall است.

نحوه‌ی عملکرد این تابع نیز به این صورت است که با توجه به ورودی‌ها تعداد True Positive، True Negative، False Positive و False Negative را به دست می‌آورد. سپس با توجه به فرمول‌های اسلایدها، معیار خواسته شده را خروجی می‌دهد.

## معیارهای ارزیابی برای Naïve Bayes

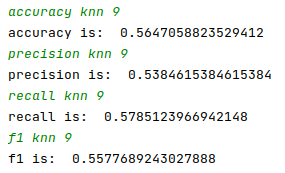
در این قسمت، با وارد کردن دستور [metric] naïve\_bayes در کنسول، می‌توان مقدار معیار ورودی را برای دسته‌بند Naïve-Bayes مشاهده کرد.



تصویر 19: مقدار معیارهای ارزیابی برای دسته‌بند Naïve Bayes

## معیارهای ارزیابی برای k-NN

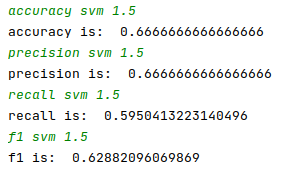
با وارد کردن دستور [metric] knn k می‌توان مقادیر معیار ورودی را برای دسته‌بند k-NN مشاهده کرد. با توجه به این که در این بخش معیارها برای بهترین k خواسته شده بود، k را برابر با 9 قرار می‌دهیم.



تصویر 20: مقدار معیارهای ارزیابی برای دسته‌بند k-NN (k = 9)

## معیارهای ارزیابی برای SVM

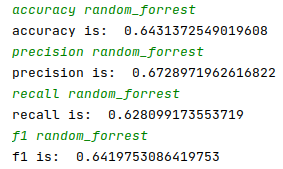
دستور [metric] svm c را در کنسول وارد می‌کنیم که با توجه به صورت پروژه، مقدار c را برابر با بهترین C، یعنی 1.5 قرار می‌دهیم.



تصویر 21: مقدار معیارهای ارزیابی برای دسته‌بند SVM (C = 1.5)

## معیارهای ارزیابی برای Random Forest

از دستور [metric] random\_forrest استفاده می‌کنیم که مقدار معیار ورودی را تحت دسته‌بندی Random Forest محاسبه می‌کند.



تصویر 22: مقدار معیارهای ارزیابی برای دسته‌بند Random Forest

## نتیجه‌گیری نهایی معیارهای ارزیابی

با توجه به تصاویر [19](#i19)، [20](#i20)، [21](#i21) و [22](#i22) می‌توان نتیجه گرفت که:

* دسته‌بند SVM بیشترین accuracy را داشته‌است.
* دسته‌بند Random Forest بیشترین مقدار precision را حاصل کرده‌است.
* دسته‌بند Naïve Bayes بیشترین مقدار F1 و recall را داشته‌است.
* معیارهای ارزیابی دسته‌بند k-NN نسبت به سایر دسته‌بندها پایین‌تر بوده‌است.

# نحوه‌ی تقسیم وظایف

وظایف اختصاص یافته به هر فرد به شرح زیر بود:

## نیما جمالی

1- پیش‌‌پردازش داده‌ها و پیاده‌سازی روش ntn

2- ساخت دو مدل فضای برداری و توابع مربوط به آنها

3- پیاده‌سازی دسته‌بند k-NN و توابع جانبی آن

4- پیاده‌سازی تابع محاسبه‌ی بهترین پارامتر k و ساختن مجموعه‌ی Validation

## سپهر فعلی

1- طراحی دسته‌بند Naïve-Bayes و توابع مربوط به آن

2- طراحی توابع محاسبه‌ی معیارهای ارزیابی

3- مقایسه‌ی معیارهای ارزیابی برای دسته‌بندها

## سینا کاظمی

1- طراحی دسته‌بندهای SVM و Random Forest

2- پیاده‌سازی تابع محاسبه‌ی بهترین پارامتر C برای دسته‌بند SVM

3- طراحی تابع اجماع نظر دسته‌بندها

4- طراحی تابع پاسخ پرسمان، بسته به دسته‌ی ورودی

# مراجع

[1] <https://scikit-learn.org/stable/modules/svm.html>

[2] <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>