

بہ نام خدا



عنوان: پروژہ اول درس ہوش محاسباتی

استاد: حسین کارشناس

اعضای گروہ:

علی پور قیصری

طاہا داوری

محمد امین مولوی زاده

## مقدمه:

در این پروژه، با استفاده از الگوریتم ژنتیک، یک سیستم استنتاج فازی برای جداسازی داده‌های متنی اسپم و غیراسپم پیاده سازی شده است. در این پروژه از مجموعه داده SMSSpamCollection از پایگاه UCI استفاده شده است که شامل 5574 داده متنی می‌باشد.

## روش انجام آزمایش:

برای انجام این پروژه، ابتدا مجموعه داده را خوانده و آن را به داده‌های آزمایش و تست به نسبت یک سوم تقسیم می‌کنیم. سپس تعدادی قانون فازی با استفاده از توابعی در ادامه توضیح داده خواهد شد می‌سازیم. سپس این مجموعه قانون را به الگوریتم ژنتیکی خود می‌دهیم که در آنجا بهینه‌ترین قوانین انتخاب می‌شوند. در الگوریتم تکاملی، کروموزوم ما نشان دهنده یک قانون می‌باشد. سپس پایگاه قانون ما که همان مجموعه والدین الگوریتم تکاملی می‌باشد، به ما بازگردانده می‌شود تا با استفاده از آنها، کلاس بندی را انجام دهیم.

در زیر کلاس‌ها و توابع به صورت مختصر توضیح داده شده است.

## کلاس data\_preprocessing:

این تابع مخصوص استخراج ویژگی‌های ایمیل‌های موجود در دیتاست داده شده می‌باشد. توابع موجود در این کلاس از قبل پیاده سازی شده بود و در اختیار ما قرار گرفت.

## کلاس Fuzzy\_functions:

این کلاس برای پیاده‌سازی توابع عضویت فازی و تست قواعد فازی است.

## تابع test:

یک لیست از قواعد را به همراه داده‌های تست دریافت می‌کند و برای هر داده تست، تطبیق آن با قواعد را محاسبه کرده و برای هر کدام از دو کلاس، جمع تطبیق‌های تمام قواعد این کلاس را محاسبه می‌کند. در نهایت، این تابع برای هر داده تست، برچسب کلاسی را باز می‌گرداند که جمع تطبیق‌های آن به کلاس صفر بیشتر باشد یا به کلاس یک. سپس لیست برچسب‌ها برگردانده می‌شود که این لیست با لیست برچسب‌های اصلی مقایسه شده و دقت ما به دست می‌آید.

## تابع `calculate\_matching`:

برای هر قاعده و یک داده، تطبیق آن داده با قاعده را محاسبه می‌کند. در این تابع از قانون ضرب درجه عضویت استفاده شده است. البته می‌توان به جای ضرب از مینیمم هم استفاده کرد. سپس ورودی ما به تابع مورد نظر داده شده و درجه عضویت کل محاسبه می‌شود.

## توابع `trapezius`، `gaussian`، `sigmoid` و `triangular`:

به ترتیب برای محاسبه مقدار عضویت فازی از توابع سیگموئیدی، گوسی، تراپزوئیدی و مثلثی استفاده می‌کنند. در این جا فرمول‌های مجموعه‌های فازی سیگموئید، فازی گوسی، فازی مثلثی متساوی الساقین، فازی دوزنقه‌های قائم الزاویه طبق داک نوشته است. علاوه بر این برای فهم بهتر فرمول‌ها توضیح هم داده شده‌اند. در هر تابع، `x` مقدار داده، `s` پهنای منحنی عضویت و `m` نقطه میانگین را دریافت کرده و مقدار عضویت فازی را محاسبه می‌کنند.

## کلاس Rule:

به طور خلاصه این کلاس برای تولید قوانین فازی ساخته شده است. در این کلاس، در ابتدا با فراخوانی متد `\_\_init\_\_` قوانین به صورت اولیه تولید می‌شود. در این گام، با استفاده از متغیرهای ورودی `maximum\_value` و `minimum\_value`، با فراخوانی متد `generate\_if\_term`، شرط‌های `if` قوانین به صورت تصادفی تولید می‌شود که تعداد آن‌ها بین ۱ تا ۵ قرار دارد. در ادامه با فراخوانی متد `generate\_class\_label`، لیبل کلاس نیز به صورت تصادفی تولید می‌شود. هر قانون دارای ویژگی `fitness` نیز است که برای محاسبه با داده‌های آموزشی مورد استفاده قرار می‌گیرد. در صورتی که قانون در حال تولید فرزندان جدید باشد، مقدار `True` دریافت می‌کند. البته مقدار پیش فرض `is\_offspring` برابر `False` است.

## کلاس genetic\_algorithm:

این کلاس یک الگوریتم ژنتیک برای یادگیری قوانین فازی برای دسته‌بندی داده‌ها استفاده می‌کند. در این الگوریتم، پارامترهایی مانند تعداد نسل‌ها، ضریب جابجایی و نرخ جهش برای بهتر شدن عملکرد الگوریتم تعیین شده‌اند. هدف این کلاس ایجاد یک مجموعه قانون فازی با استفاده از محاسبات ژنتیکی و داده‌های آموزشی است. در این روش، مجموعه‌ای از قوانین فازی به صورت تصادفی ایجاد می‌شود و سپس قوانین با دقت بهتر و انطباق بیشتر با داده‌های آموزشی انتخاب می‌شوند. به علاوه، نرخ جهش و ضریب جابجایی برای ایجاد تغییرات در جمعیت قوانین تنظیم می‌شوند تا الگوریتم بتواند به یک حالت بهینه برای قوانین فازی برای دسته‌بندی داده‌ها برسد.

کلاس genetic\_algorithm از توابع max\_min\_for\_initial\_generation، selection، crossover، mutation، mutation\_if\_term و fitness تشکیل شده است. این توابع به طور مختصر در ادامه توضیح داده شده است.

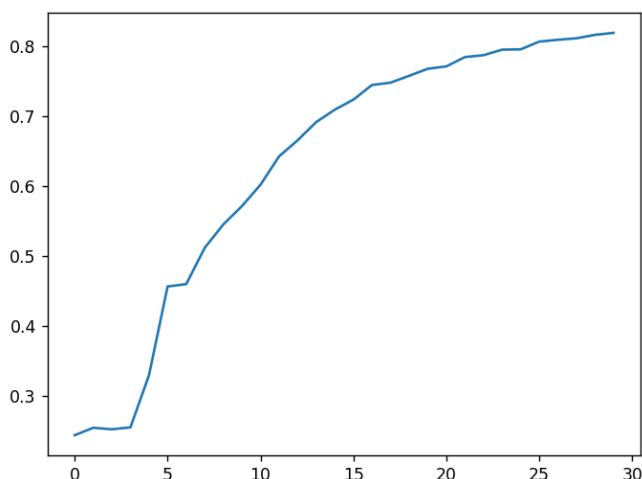
- تابع `max_min_for_initial_generation` مشخص کننده بیشینه و کمینه مقادیر مجاز برای مولفه‌های قاعده را محاسبه می‌کند.
- سپس تابع `algorithm` با استفاده از این پارامترها، الگوریتم ژنتیک را بر روی والدین اجرا می‌کند.
- تابع `selection` والدینی را که بیشترین `fitness` را دارند را انتخاب می‌کند.
- تابع `crossover` برای هر جفت والد، دو فرزند جدید ایجاد می‌کند که قاعده‌های والدین را یاد می‌گیرند.
- تابع `mutation` قواعد فرزندان را با احتمال مشخص شده جهش می‌دهد.
- در نهایت، این کلاس شامل یک چند جمله‌ای سازنده است که با ورود داده‌های آموزش `X_train` و `y_train`، جمعیت و سایر پارامترها، مقدار `fitness_score` و `parent_pool` را برمی‌گرداند.

## نتایج به دست آمده:

نتایج به دست آمده به صورت زیر می باشد.

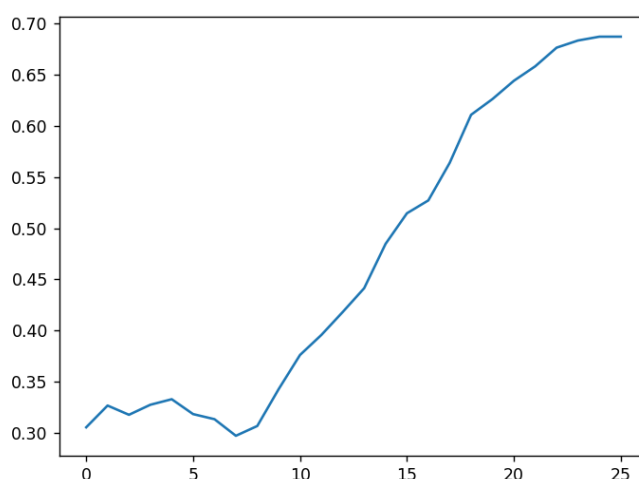
نمودار و داده های به دست آمده با  $\text{crossover rate, mutation rate} = 0.1$

```
generation 0 average fitness: 0.24429013043659872
generation 1 average fitness: 0.25497627424211666
generation 2 average fitness: 0.25277321317426155
generation 3 average fitness: 0.2555396542393686
generation 4 average fitness: 0.33012978455015274
generation 5 average fitness: 0.4568081825084991
generation 6 average fitness: 0.46011078887040535
generation 7 average fitness: 0.5120818093198554
generation 8 average fitness: 0.5454489536314719
generation 9 average fitness: 0.5715541331494532
generation 10 average fitness: 0.6021410116367801
generation 11 average fitness: 0.6426761409645819
generation 12 average fitness: 0.6656704594278361
generation 13 average fitness: 0.6916856426451244
generation 14 average fitness: 0.7093906731688446
generation 15 average fitness: 0.7237212909711732
generation 16 average fitness: 0.7446288171323358
generation 17 average fitness: 0.7479179719081139
generation 18 average fitness: 0.7577415640286008
generation 19 average fitness: 0.7677768529043969
generation 20 average fitness: 0.7711974237323056
generation 21 average fitness: 0.7844716339779183
generation 22 average fitness: 0.787159537258129
generation 23 average fitness: 0.7950703141963592
generation 24 average fitness: 0.7955123482793357
generation 25 average fitness: 0.8065872560169374
generation 26 average fitness: 0.8091850161710014
generation 27 average fitness: 0.8113023509525809
generation 28 average fitness: 0.8162108902863807
generation 29 average fitness: 0.819089978758442
accuracy is: 0.7695652173913043
```



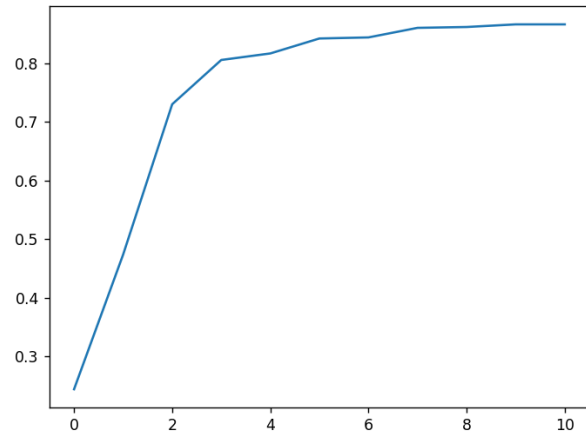
نمودار و داده های به دست آمده با  $\text{crossover rate} = 0.1, \text{mutation rate} = 0.9$

```
generation 0 average fitness: 0.3056093419154676
generation 1 average fitness: 0.3268516587918112
generation 2 average fitness: 0.3178332641473493
generation 3 average fitness: 0.3276196858490817
generation 4 average fitness: 0.33303547852412385
generation 5 average fitness: 0.31850365436548544
generation 6 average fitness: 0.31353725080599276
generation 7 average fitness: 0.2973398270677378
generation 8 average fitness: 0.3068771158411464
generation 9 average fitness: 0.3428625935396114
generation 10 average fitness: 0.37624721220285606
generation 11 average fitness: 0.39581350224752854
generation 12 average fitness: 0.4182654524082412
generation 13 average fitness: 0.4414277876233473
generation 14 average fitness: 0.4847039604653858
generation 15 average fitness: 0.5145941245267123
generation 16 average fitness: 0.5271412623520085
generation 17 average fitness: 0.5638387443448114
generation 18 average fitness: 0.6107721369372532
generation 19 average fitness: 0.6261062828215457
generation 20 average fitness: 0.6439481084794438
generation 21 average fitness: 0.6580987597191704
generation 22 average fitness: 0.6763092409093449
generation 23 average fitness: 0.6832758047253568
generation 24 average fitness: 0.6870206754352742
generation 25 average fitness: 0.6870206754352742
accuracy is: 0.8597826086956522
```



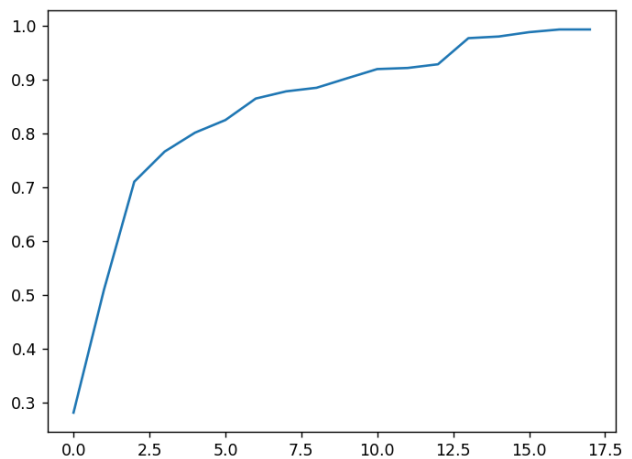
نمودار و داده های به دست آمده با  $\text{crossover rate} = 0.9$ ,  $\text{mutation rate} = 0.1$

```
generation 0 average fitness: 0.24341509828875935
generation 1 average fitness: 0.4734763816935918
generation 2 average fitness: 0.7304032015396871
generation 3 average fitness: 0.8060604793823528
generation 4 average fitness: 0.8172328332794495
generation 5 average fitness: 0.8427991422598263
generation 6 average fitness: 0.8446191578713312
generation 7 average fitness: 0.8609725190511132
generation 8 average fitness: 0.8624213699665839
generation 9 average fitness: 0.8669115341983153
generation 10 average fitness: 0.8669115341983153
accuracy is: 0.8315217391304348
```



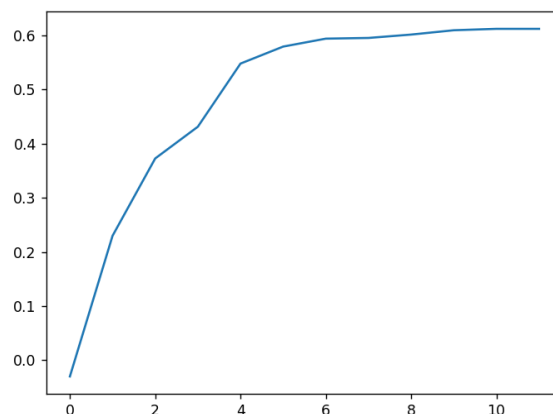
نمودار و داده های به دست آمده با  $\text{crossover rate, mutation rate} = 0.9$

```
generation 0 average fitness: 0.2809499091181227
generation 1 average fitness: 0.5089864148800625
generation 2 average fitness: 0.710433885191662
generation 3 average fitness: 0.7664101340358519
generation 4 average fitness: 0.8016133583171714
generation 5 average fitness: 0.8250442638155147
generation 6 average fitness: 0.8650440642485292
generation 7 average fitness: 0.8784087259949334
generation 8 average fitness: 0.8850335272864506
generation 9 average fitness: 0.9025835163742397
generation 10 average fitness: 0.9198501546968585
generation 11 average fitness: 0.92187392026137
generation 12 average fitness: 0.92879022813305
generation 13 average fitness: 0.9773266836706003
generation 14 average fitness: 0.980373942528692
generation 15 average fitness: 0.9885110020592239
generation 16 average fitness: 0.9934966462069116
generation 17 average fitness: 0.9934966462069116
accuracy is: 0.8510869565217392
```



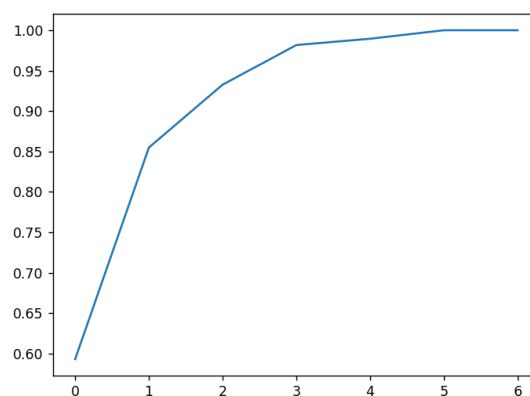
نمودار و داده های به دست آمده برای استفاده از min به جای ضرب برای درجه عضویت

```
generation 1 average fitness: 0.22978549119398772
generation 2 average fitness: 0.3725694812263465
generation 3 average fitness: 0.43118256207331707
generation 4 average fitness: 0.5477470756292588
generation 5 average fitness: 0.5791323471272302
generation 6 average fitness: 0.5937994928561001
generation 7 average fitness: 0.5950908901675747
generation 8 average fitness: 0.6013134294406641
generation 9 average fitness: 0.6092997204465134
generation 10 average fitness: 0.6119116955730792
generation 11 average fitness: 0.6119116955730792
accuracy is: 0.8771739130434782
```



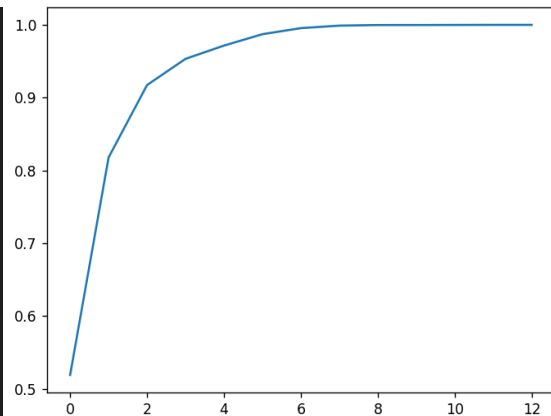
نمودار و داده های به دست آمده با اندازه پایگاه قانون 50 تایی

```
generation 0 average fitness: 0.5929992486751502
generation 1 average fitness: 0.854880149454944
generation 2 average fitness: 0.9326738348743904
generation 3 average fitness: 0.9817100973162282
generation 4 average fitness: 0.9895010321322646
generation 5 average fitness: 1.0
generation 6 average fitness: 1.0
accuracy is: 0.8016304347826086
```



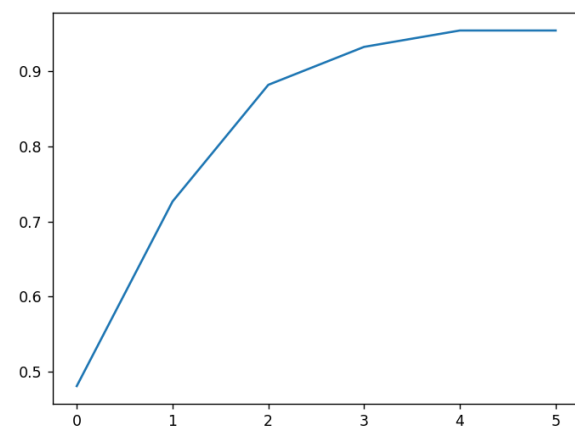
نمودار و داده های به دست آمده با اندازه پایگاه قانون 100 تایی

```
generation 0 average fitness: 0.5191087507228977
generation 1 average fitness: 0.8178942426053128
generation 2 average fitness: 0.9173306985743547
generation 3 average fitness: 0.9533254472125888
generation 4 average fitness: 0.9715209944004798
generation 5 average fitness: 0.9871913131829704
generation 6 average fitness: 0.9955061426273024
generation 7 average fitness: 0.9989326673607393
generation 8 average fitness: 0.9997385653974833
generation 9 average fitness: 0.9997688937065582
generation 10 average fitness: 0.9999004424864688
generation 11 average fitness: 1.0
generation 12 average fitness: 1.0
accuracy is: 0.8586956521739131
```



نمودار و داده های به دست آمده با اندازه پایگاه قانون 500 تایی

```
generation 0 average fitness: 0.48106579438993047
generation 1 average fitness: 0.7265628455261124
generation 2 average fitness: 0.8817892362322227
generation 3 average fitness: 0.9322623284934066
generation 4 average fitness: 0.9540674943110029
generation 5 average fitness: 0.9540674943110029
accuracy is: 0.8690217391304348
```





### تحلیل نتایج:

برای داده‌های به دست آمده از crossover rate و mutation rate به این نتیجه می‌رسیم که با تغییرات این دو به نسبت خاص و خوبی می‌توان دقت را افزایش داد. میزان crossover rate بیشتر و mutation rate کمتر به ما دقت بهینه را می‌دهد. کمینه گرفتن به جای ضرب در محاسبه درجه عضویت، دقتی کمی بهتری با توجه به اشکال به ما می‌دهد. هرچه تعداد قانون‌های مجموعه قانون‌فازی بیشتر باشد، دقت بالاتر می‌رود ولی باید توجه کرد که زمان محاسبه افزایش می‌یابد.

### پیشنهاد:

با توجه به مدت زمان طولانی بودن انتخاب ویژگی (feature selection) می‌توان با یکبار خواندن اجرای آن و ذخیره در فایل، دفعات بعد بدون معطلی ویژگی‌ها را خواند. اگر مجموعه داده شامل تعداد یکسانی از داده‌های متنی اسپم و غیراسپم باشد. کار ما در انجام الگوریتم راحت‌تر می‌باشد. شاید بتوان برای در نظر گرفتن قوانین اولیه یکسری شروطی در نظر گرفت که بهینه‌تر باشد. شاید نرخ جهش فعلی که ثابت است مناسب الگوریتم نباشد، بنابراین می‌توان از نرخ جهش پویا استفاده کرد که بر اساس برآزندگی قوانین نسل‌های قبل به دست آمده باشد.

### منابع:

[/https://chat.openai.com/](https://chat.openai.com/)

[/https://treatta.com/genetic-algorithms-introductio](https://treatta.com/genetic-algorithms-introductio)

[Machine Intelligence - Lecture 18 \(Evolutionary Algorithms\) - YouTube](#)