**Advanced Intelligent Information Systems – Data Mining**

Is it Possible to classify the popularity of a project using metrics attained from GitHub?

-Jordan McDonald-

**1. Application Problem**

GitHub is a web based GIT repository service that enables version control and source management for teams (distributed or local) to collaborate in software development. GitHub is a platform which thrives upon open source development as well as supporting business applications, at the time of writing GitHub reports having 14 million users and 35 million repositories making it the largest host of source code in the world. This represents a period of rapid growth considering in 2010, announced on the official GitHub blog it was revealed that one million repositories were hosted on GitHub. In the context of data mining this is an application context that is ripe for investigation and has inspired this study.

The problem this report will tackle will be – is it possible to classify the popularity of a repository using only projects that can from obtained from a) the GitHub website b) the GitHub API. GitHub presents a metric called ‘stargazers’ which represents a user who has shown an interest in the project, therefore the more popular a repository is, the higher the amount of stargazers present so this will act as the class attribute which requires a decision. The variance of popularity needs to be quantified in order for this study to succeed, to do this differing sets of stargazers have been devised which provide a multi class range of potential classifications, see below for the ranges.

* <500
* 500-2000
* 2000-5000
* 5000-10000
* >10000

To achieve this goal of classification consideration needs to be given to the types of metrics which can be extracted, from either the API or the website itself, refer to figure one for the metrics that will be utilized and what each mean in the context of the GitHub platform (each metric represents an attribute in the dataset). To ensure the languages selected are not arbitrary, the most popular on the platform have been chosen [1], from this each will be split into subsets according to the various stargazers class attribute ranges.

|  |  |
| --- | --- |
| **Metric** | **Description** |
| Language | This will be a multi-language study and this will represent what programming language is in the instance. |
| Commit | A snapshot of the repository |
| Branches | A separate version of the repository which enables experimentation from outside the master branch. |
| Watchers | Users who subscribe to a repository and receive updates. |
| Forks | Similar to a branch by allowing users to clone the repository and work on it independently. |
| Issues | Users can raise issues when bugs are discovered/suggest areas for improvement. |
| Releases | Represents a version of the system that has been released to the public. |
| LOC | Total lines of code in the repository |
| Pull Requests | A request to merge a change to the master branch |
| Read me? | A file explaining the repository, in this case, does one exist? |
| <100 contributors? | Used to check the size of the development team and see how that impacts popularity. |
| Commits in the last month? | Used to assist in determining the activity rate of the contributors. |
| Stars | Allows a user to register interest in a project |

Figure 1 – metrics used in this study as attributes.

**A) Dataset Generation Strategy**

Now it would be prudent to show how the data will be collected, GitHub provides an advanced search engine (https://github.com/search/advanced) which allows a user to discover repositories that meet a series of parameters. This was utilized to filter the selection by language and then by then the potential class attribute values, see below for an example search query.

* language:C stars:500..2000 (finds the projects which have 500-200 stars for the C language).

This would result in a list of options that could be selected as part of the dataset, to ensure that no bias was applied the first results were chosen indiscriminately. From this the repository could be visited which presents most of the data required in a visual manner, however in some cases API access will be required. To enable this process the ‘Darwin’ workbench has been leveraged, I developed this as part of my dissertation and decided it would be wise to apply it where required to this scenario, see figure three to view the architecture of this system. The project selection strategy should also be documented, see figure two for the general approach taken. It should also be noted that some languages could not fulfill the class attribute data ranges (e.g. > 10000 stars, CSS had no repositories meeting this requirement) in this case the attributes were padded by adding more to the other ranges where possible.

Identify the top ten languages

Filter the repositories for each language using the search engine

If 12 selected move onto the next language

Select 2-3 repositories for each class attribute range

GitHub API

Webpage

URL(s)

JSON data

Java Servlet

JSON extractor module (JS)

Raw data

MongoDB

DB Query

R Environment

Figure 3 – Darwin system architecture

It should also be considered that this report will use real world data that will be original and extracted by myself, this leads a significant amount of time spent forming the dataset which should be considered when discussing the significance and technical difficulty of the report. This will lead to a reduced scope in the number of instances that are use (120 total) but I feel the originality of the study (which to my knowledge has not been tackled using the two classification techniques chosen) outweighs that restriction.

**B) Description of the dataset**

The dataset will be formatted as an attribute relation file format (arff) and as mentioned prior has been generated using real world repository data. The dataset has been attached in the appendix for future reference. The bulk of the data will be numerical and typically represents counts for a certain metric, so in the case of commits the value for the instance indicates the total amount for that particular repository. Nominal/categorical data is present for three of the attributes with the potential values of ‘Yes’ or ‘No’ which indicate a binary outcome. The class attribute uses ordinal data be presenting various ordered ranges of values with the first and last outcomes being more general using either ‘<’ or ‘>’ to capture the remaining repositories that do not fit within any of the set ranges.

**2. Machine learning Technique & Classification Models**

Now it is crucial to describe in detail the machine learning technique that will be leveraged as well as the algorithms that will be utilized to enable the technique. Classification has been chosen which can use unsupervised learning (class and labels of the data are unknown, involves clustering) or in the case of this reports scenario supervised learning (the data is labelled with pre-determined classes). This is a reflection of the dataset and application problem which compliments supervised learning since the data is freely available from either the GitHub API or the repository details on the website, so this was a decision driven from analyzing the data and the domain. Supervised learning typically has two steps 1) learning – uses the training data to generate a classifier 2) testing – passing new data into the classifier to assess the models performance and classify based on the training model. See figure four for the general classification process which this study will follow.

Training Data

Results

10 fold cross validation

Learning Algorithm

* The training sample is partitioned into k equal sized subsets
* One of the subsets is retained as the validation set and the other k-1sets are the training data
* The process is repeated k times with each subset used as the validation data once
* The results can then be averaged to produce a single estimation

Figure 4 – classification process for this report

**A) Machine learning tool**

In order to apply the machine learning techniques required for this study it was crucial to select a tool which supported varying algorithms and provided flexibility in the dataset chosen (even more so since this reports data is based from the real world and created from scratch). Therefore Weka was selected, Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization which fits the key requirements. Initially the dataset will be loaded into Weka via the ‘open file’ functionality provided as an ARFF file type, from here it is possible to preprocess the data or select classification tab to apply a series of machine learning techniques. It should be noted that Weka is written in Java but I will be focusing on the graphical user interface provided.

**B) Classification techniques – (Chosen based on the lecture material)**

J48 decision tree – known as a statistical classifier J48 is a Java implementation of the 4.5 decision tree algorithm developed by Ross Quinlan. In terms of structure the nodes of a decision tree denote training data attributes, the branches between nodes inform the potential values from a node and the terminal node tells us the final classification of a specific instance. A key component of this algorithm is the formation of the tree using the training data, particular how to choose the attributes which represent a split, see below for pseudo code of the approach this algorithm utilizes.

1. Check for the base cases
2. For each attribute

* Identify the attribute which discriminates the data most clearly (highest information gain)

3. Create a decision node that splits on the best attribute

4. Recursively repeat for the next nodes until all the data is distributed

Base Cases [4]

1. All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class
2. None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class
3. Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value

Naïve Bayes – In order the ensure the comparison between the two classification algorithms was interesting and significant a different approach from decision trees was required, in this case naïve Bayes was selected which focusing on probabilities to give predictions (see figure five for the basic formula). Built upon Bayes theorem this algorithm has the assumption the of independence between every pair of features, which in this case is a weakness as it is very possible the amount of stargazers is directly driven by the repositories activity such as commits, however this is opinion based with no empirical data so the selection of Naïve Bayes is still valid.



Figure five – General concept behind the naive Bayes approach

Now it would be prudent to discuss how the algorithm operates in generating probabilistic predictions for a class attribute based on the attributes features…

1) Compute the probabilities of all of the attributes values in the relation against the class attributes (i.e. P(readme=’yes’|stars=<500) = 13/24 = 0.5416) – repeat for each class attribute option.

2) Find the maximum probably that an attribute will belong to a class and select that as the prediction (i.e. for read me value ‘yes’ 13/24 \* 1/24 \* 5/24 \* 2/24 \*2\*24 = 0.000065 - then repeat for read me value ‘no’)

3) Whatever case has the highest probability is the one selected as the prediction.

**C) Classification Models**

J48 – Initially we will consider this algorithm, it is possible to tweak parameters in order to vary the behavior of the decision tree. These include the confidence factor for pruning, by default it is 0.25, in addition to this the minimum number of objects can be specified. The tree configuration will follow the Weka default, see below for some of the details involved with this tree, empirically the default tended to perform best which reinforces the decision.

* Confidence factor = 0.25
* minNumObj = 2
* numFolds = 3
* seed = 1

See figure 6 below for a visual representation of the J48 tree (and the appendix for a textual representation), in addition to this we should consider the size of the tree (8) and the number of leaves (15). It also took below a second to generate the model so the efficiency of the training process is very efficient in this size of dataset, however this may vary when a much larger set of instances are supplied. We can also see that not all of the attributes are present in the decision tree which indicates that these attributes did not have ‘win’ any of the splits that occurred before all the instances were classified and they may have less significance in the context of J48 compared to the other selected nodes.

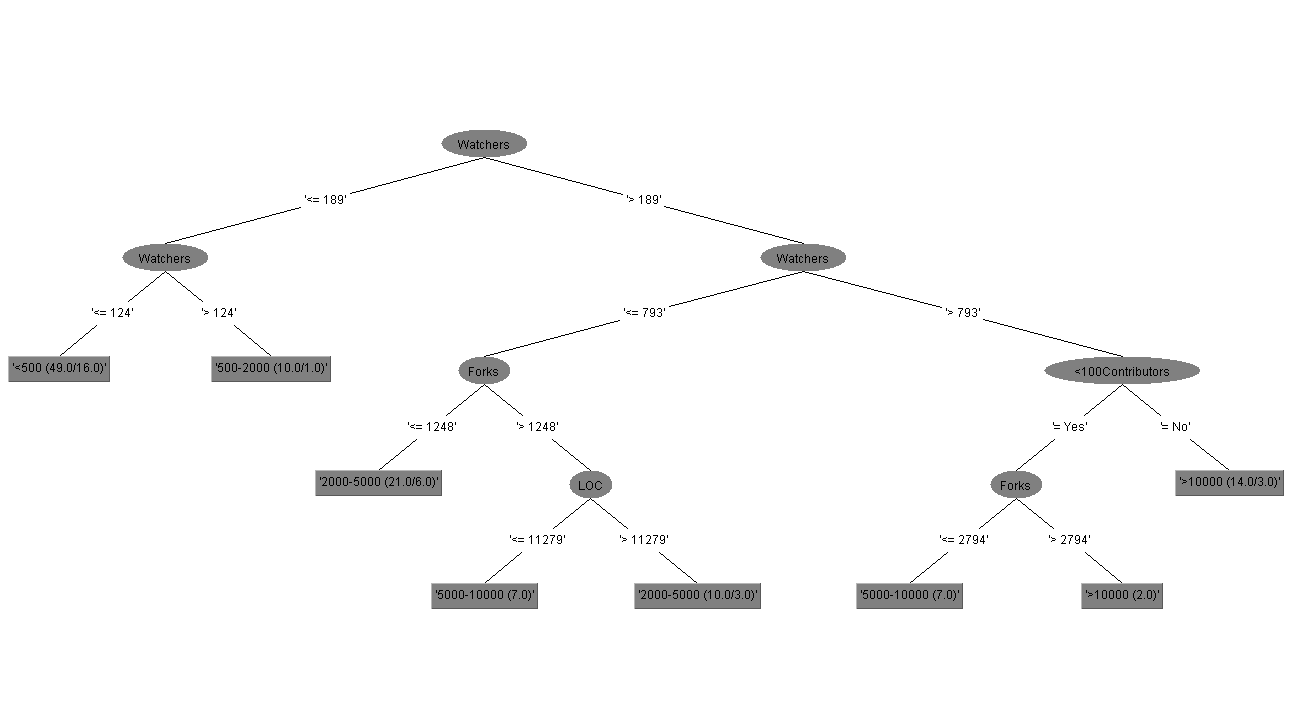


Figure 6 – visual representation of the tree model

Naïve Bayes – This algorithm works effectively by initially creating a frequency table for the categorical values, an example of one of these tables has been given in figure 7 which shows the distribution of an attribute values against the class attribute. In addition to this numerical values will then be applied to various calculations such as the mean and standard deviation for the values under the umbrella of the class attribute in question, see figure 8 for an example of this.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | <500 | 500-2000 | 2000-5000 | 5000-10000 | >10000 |
| Yes | 34 | 34 | 23 | 19 | 15 |
| No | 1 | 1 | 1 | 1 | 1 |
| Total | 35 | 35 | 23 | 20 | 16 |

Figure 7 – frequency table for the ‘readme’ attribute

From this table some observations can be made, it is clear that the majority of the repositories utilize a ‘read me’ file to inform the visitors. This one sided distribution could be the reason this attribute does not have a presence in the decision tree. As in the J48 implementation the efficiency of the model generation was very high and in datasets of this size can created quickly but this may not hold for larger datasets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | <500 | 500-2000 | 2000-5000 | 5000-10000 | >10000 |
| Mean | 96.4187 | 72.314 | 25.8264 | 309.3434 | 48.7013 |
| Std. dev | 383.7022 | 353.622 | 58.668 | 1139.3934 | 119.2993 |
| Weight sum | 33 | 32 | 22 | 18 | 14 |
| precision | 113.6364 | 113.6364 | 113.6364 | 113.6364 | 113.6364 |

Figure 8 – statistics for the ‘pull request’ metric generated in the model

A limitation in this algorithm is the assumption that all data is categorical, however in this dataset the majority of the data is numerical. Luckily Weka handles this occurrence by applying Gaussian distributions by default for numeric attributes. However, it also has options to use supervised discretization or kernel density estimation. For this study will use supervised discretization (as suggested in the lecture notes) and with the knowledge of inspecting the dataset it is clear the distribution of the bulk of the numerical attributes are not within a Gaussian distribution.

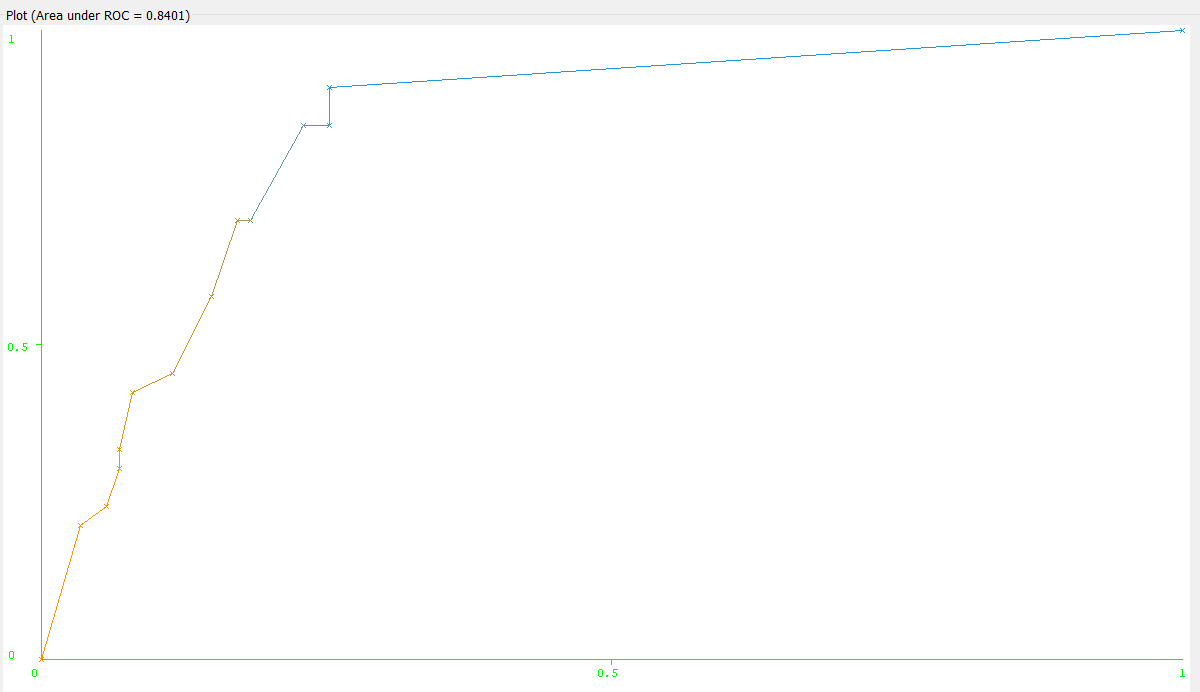
**3. Evaluation**

Initially we will evaluate the J48 results obtained from running cross validation on the training dataset and discuss how well the application problem has been tackled in this context. The overall accuracy of this algorithm was 57.5% which on the surface value is very poor showing little affinity to correct classifications that exceed the performance of a random classifier (discussion will be made later why this occurs). See figure 9 for a breakdown of the results obtained from Weka.

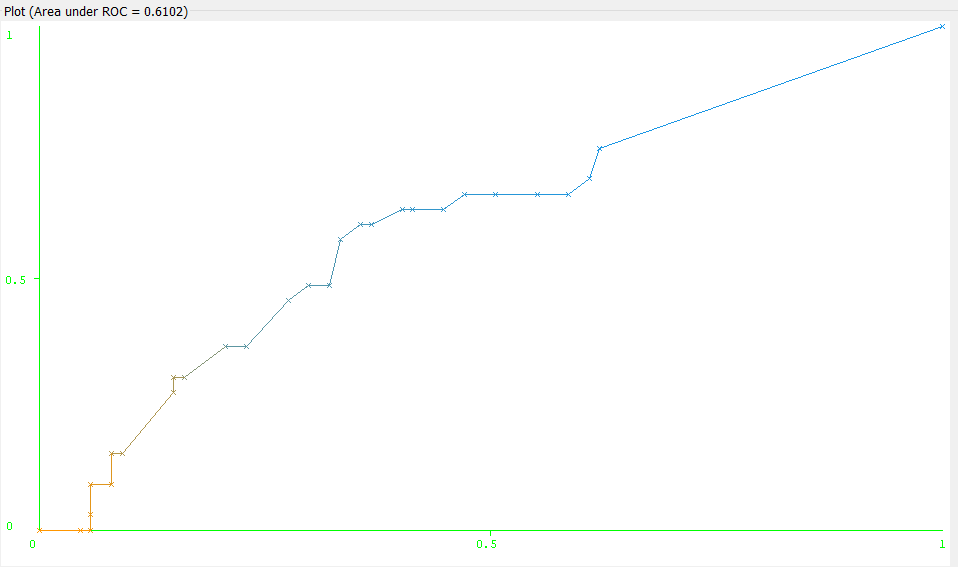
|  |  |
| --- | --- |
| Correctly Classified instances | 57.5 |
| Kappa statistic | 0.4574 |
| Mean absolute error | 0.1943 |
| Root mean squared error | 0.3711 |
| Relative absolute error | 62.2342 |
| Root relative squared error | 93.9158 |
| Precision (weighted average) | 0.566 |
| Recall (weighted average) | 0.575 |
| F-Measure (weighted average) | 0.563 |

Figure 9 – Weka feedback metrics

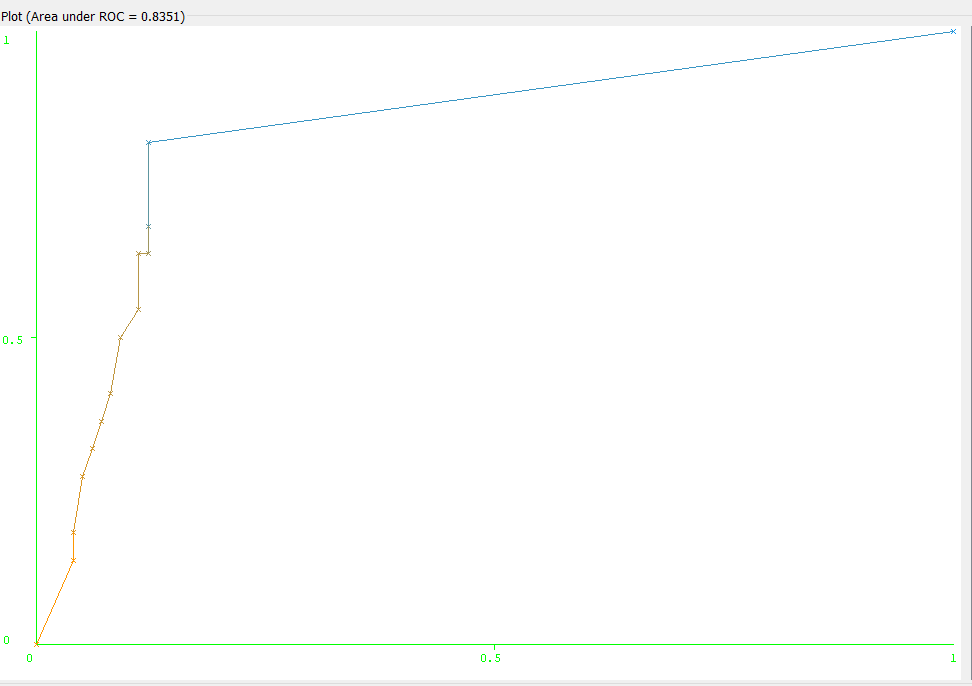
The Kappa statistic takes into account random chance (which is ideal since our accuracy is almost 50/50) and deals with observed and expected accuracies. [5] Characterize the Kappa statistic magnitude and state that 0.21–0.40 as fair agreement, however this is not universally accepted. The host of error statistic typically compare the true values to the estimates and ‘how far away’ there are from one another, in terms of MAE and RMSE the correlations attained are quite weak and the error rates for RAE and RRSE are quite high which reinforces the poor overall accuracy that has been achieved by this classifier. The precision, recall and f-measure reflect the overall accuracy as well. It is also a good option to consider the results not only for the overall process but for each class attribute, to do this ROC curves have been leveraged (figure 10) which plots the true positive rate and false positive rate.



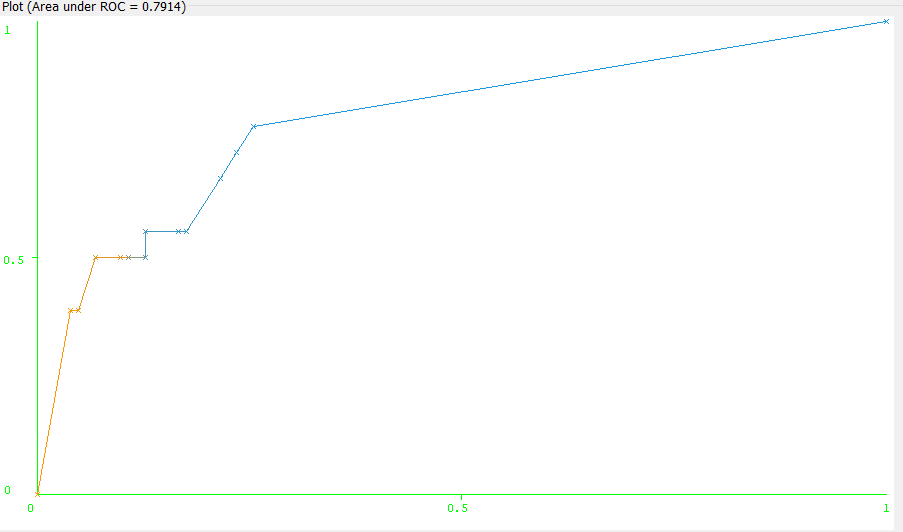
<500 class - AUR = 0.8401



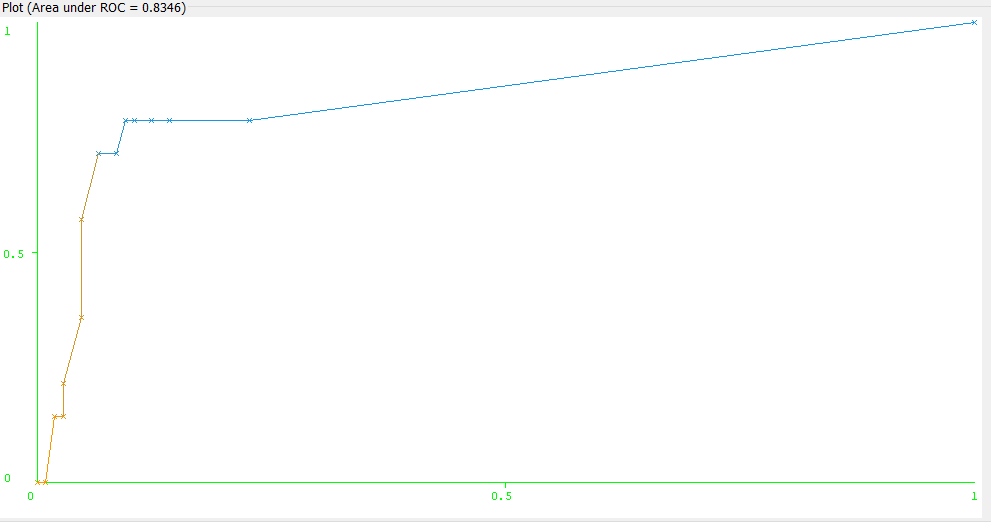
500-2000 class – AUR = 0.62



2000-5000 class – 0.817 AUR



5000-10000 class – 0.7783 AUR



>10000 – 0.8346 AUR

Figure 10 – ROC curves for each class attribute for J48 in order from <500…>10000

It should be considered the interpretation of these ROC curves, a higher area under curve and curve that remains closer to the left hand margin graph margin typically represents better accuracy and gives the overall tradeoff between specificity and sensitivity. The 500-2000 class attribute shows the worst performance in terms of accuracy which indicates that repositories within this popularity threshold show a high level of variance (which leads to them being classified incorrectly as other attributes), this is reflected upon manual inspection of the dataset with no general trend or pattern in the metrics attained which explains why it is commonly misclassified. The other class attributes perform much better than the latter and therefore can distinguish between the binary cases of a) part of this class b) not part of this class much clearer, however despite the AUR being relatively good this a multi class classification problem not a binary one so this ability to distinguish between two cases may have less significance. See below for the confusion matrix to see the distribution of classifications.

a b c d e <-- classified as

25 8 0 0 0 | a = <500

13 11 7 1 1 | b = 500-2000

0 2 16 4 0 | c = 2000-5000

0 0 5 9 4 | d = 5000-10000

0 1 0 5 8 | e = >10000

This reinforces the previous observations and highlights the variance of results in class b which has a presence in every classification category. A useful trend to notice (aside from class b) is that as the popularity increases the less presence the more popular class will have in less popular classes, which indicates that the metrics associated with popular projects correlate with popularity. For example in class d there are no instances classified as either class a or b which reinforces the point that the overall classification shifts towards class e as popularity increases and vice versa.

The compactness of the model is also noteworthy with significant pruning occurring due to the restriction on the minimum number of objects. This has led to a portion of the attributes not having a presence in the tree which indicates that they are not required by the J48 splitting process in order to classify the instances fully. Robustness is not a consideration as no missing values or typical noise is present while interpretability to an extent reflects the domain of open source development, which is considered later.

**A) Comparison & determining the best model**

Now the second algorithm will be evaluated using 10 fold cross validation. In this section while analyzing the results independently a comparison (now that J48 provided a base line) will be made to the J48 algorithm feedback in order to determine the best performing model – for brevity the ROC curve won’t be included (just the AUR), rather the metrics themselves will form the comparison. See figure 11 below for the metrics attained from using the Bayes classifier as a predictive tool.

|  |  |  |
| --- | --- | --- |
| metric | J48 | Bayes |
| Accuracy | 57.5 | 50% |
| Kappa statistic | 0.4574 | 0.3495 |
| Mean absolute error | 0.1943 | 0.1992 |
| Root mean squared error | 0.3711 | 0.4212 |
| Relative absolute error | 62.2342 | 63.8063 % |
| Root relative squared error | 93.9158 | 106.6094 % |
| Precision | 0.566 | 0.468 |
| Recall | 0.575 | 0.5 |
| F-measure | 0.563 | 0.467 |

Figure 11 – comparison of Bayes and J48 results

If we consider accuracy it is clear that the J48 algorithm outperforms Naive Bayes by 7.5% when correct classifications are compared to the ground truth labels. This is reflected in the improved precision, recall and f-measure as well as the error metrics overall showing the potential for mistakes is reduced, albeit the difference is not hugely significant the decision tree is better in all cases. The next step will be to evaluate the ROC curve area under roc curve results for Bayes compared to J48, see figure 12.

|  |  |  |
| --- | --- | --- |
| Class attribute | J48 | Bayes |
| <500 | 0.883 | 0.882 |
| 500-2000 | 0.62 | 0.605 |
| 2000-5000 | 0.817 | 0.713 |
| 5000-10000 | 0.778 | 0.76 |
| >10000 | 0.835 | 0.758 |

Figure 12 – comparison of ROC area

The table shows that the values for the first, second and fourth class attributes remain quite similar, however the remaining two are significantly reduced in when Bayes is compared to J48. This indicates that the ability to correctly distinguish between whether an instance belongs to a lass in a binary decision suffers, which will lead to an increased amount of incorrect classifications, this is a reflection of the reduced accuracy and suggest that these classes are the points of contention where J48 outperforms Bayes. Overall the best classifier is the J48 decision tree in this context.

a b c d e <-- classified as

27 5 0 1 0 | a = <500

14 14 2 2 1 | b = 500-2000

0 14 3 2 3 | c = 2000-5000

0 6 4 4 4 | d = 5000-10000

0 1 0 1 12 | e = >10000

The confusion matrix above for the Naive Bayes results indicates a similar trend as J48 where the middle classes show a degree of overlap with the majority of attributes when they are incorrectly classified while the extreme cases appear to hold closer to the ‘true’ class. It is interesting that class C shows a significant amount of errors and appears to be the key driver of why the Bayes classifier suffers compared to J48 (which performs much better) due to the fact is the middle category it is possible that the instances show the greatest variation as they are stuck between being popular and less popular.

**B) The impact of the domain on the results**

While investigating the raw results is a worthy exercise it also crucial to consider why in both cases the classifiers show to an extent results which are not better than random, this I believe is a side effect of the chosen application problem/domain. GitHub is the home of modern OSS (open source software) which in contrast to typical development models thrives on dynamic interactions from the community in tandem with a dedicated team (in most cases). This leads to contributions such as commits, forks and branches to change unpredictably and the volume and extent of these changes are often difficult to anticipate. This form of the crux of why the techniques applied in this study have shown little success, in addition to the purpose of the repository – some fit a precise need and may have no reason to change (expected in CSS/HTML languages) whereas other require rapid alterations to meet user demand.

The edge cases need to be considered (<500 and >10000) which compared to the other possible classes encompass a more precise clear cut set of repositories. For example >10000 stargazers could actually in some instances mean five times that amount, therefore it is expected that metrics associated with that particular instance will be significant greater than other classes, which will make it easier to classify, this is generally reflected in the confusion matrices which are either correct or one category below (aside from an outlier). The inverse applies to the <500 class in which the repositories could be extremely unpopular to the extent where the metrics associated with it are many times smaller than the subsequent classes.

Finally it should be considered, are the attributes chosen really an accurate reflection of a projects popularity? To what extent are these useful predictors? In some cases it is obvious, in the case of language it is expected that as technology evolves some programming languages will gain popularity and greater use, and in [1] this is reinforced. However in other metrics such as ‘watchers’ it is not as clear cut, this typically refer to users who subscribe to every event that takes place in a repository which in a lot of cases is also the contributors, so would the typical user of GitHub really utilize this facility? That is up for debate.

**C)** Additional (Bonus) Study – Correlation

An interesting side task to consider would be, do any of the metrics in isolation correlate with stargazers? This will provide a fine grain approach to supplement the prior analysis and evaluation. To enable this the dataset has been integrated into Microsoft Excel, from this it is possible to select two data series and determine the strength of correlation, see figure 12 for the results of this process for each metric correlated against stargazers. It should be noted that attaining significance tests from Excel is not an easy process so the rraw result is fine in order to enable a discussion.

|  |  |  |
| --- | --- | --- |
| **Metric A** | **Metric B** | **Pearson product moment correlation coefficient** |
| Commits | Star Gazers | 0.53534918 |
| Branches | Star Gazers | 0.838899047 |
| Watchers | Star Gazers | 0.803562797 |
| Forks | Star Gazers | 0.838899 |
| Issues | Star Gazers | 0.551472419 |
| Releases | Star Gazers | 0.670672789 |
| LOC | Star Gazers | 0.503790773 |
| Pull Requests | Star Gazers | -0.050981072 |

Figure 14 – correlation/association between two metrics

It is clear that some of the metrics do indeed have a strong correlation with stargazers whereas some others do not. Branches, watchers and forks are the attributes which show a strong affinity to the class attribute, this could explain the poor results in the prior classification algorithms as it is possible that the lesser performing correlations are dramatically affecting the results. To test this theory I have generated a model using only the metrics which performed best in the correlations, this will help determine if the overall classification is improved rather than analyzing each step in detail as seen before (that is out of the scope of this study). See figure 15.

|  |  |  |
| --- | --- | --- |
| metric | J48 | Bayes |
| Accuracy | 58.333 | 57.5 |
| Kappa statistic | 0.4667 | 0.4455 |
| Mean absolute error | 0.1849 | 0.1769 |
| Root mean squared error | 0.3562 | 0.3675 |
| Relative absolute error | 59.208 | 56.8063 % |
| Root relative squared error | 90.1531 | 93.0204 % |
| Precision | 0.57 | 0.559 |
| Recall | 0.583 | 0.575 |
| F-measure | 0.576 | 0.55 |

We can see that using only the metrics with a high correlations does give some improvement, albeit the level of improvement is not significant enough to warrant further study. This shows that correlation is only one facet that enables accurate classification and in this case has not performed well enough to draw a connection between classification accuracy and correlation affinity, however this was a worthwhile bonus section to look into for this report.

**4. Discussion of the benefits of the data mining solution**

Over the course of the report I have discussed the application problem in varying levels of context and the significance of this study in predicting the popularity of a repository on GitHub, however a formal section like this will provide a summary. Initially novelty will be considered, general data mining in GitHub has become prominent in recent years [6][7][8] with varying approaches taken, however to my knowledge and through extensive research using different search facilities it is clear applying the classification techniques in this study to determining popularity has not yet been accomplished, therefore novelty and significance are inherently present.

Benefits of this study initially were to provide an answer to the research question ‘Is it Possible to classify the popularity of a project using metrics attained from GitHub?’ which I believe has been yielded based upon the results. The final outcome was that no, it is not possible in this study – this reflects the accuracy which in both algorithms cases show no affinity towards having an ability to classify correctly, as discussed before this is a byproduct of either the small dataset or the nature of the open source paradigm. While the outcome is not positive it offers something to the field and avenues for future study.

Further novelty to the study was to only use metrics that can be attained from the GitHub platform whether the site or the API. It was of interest to discover if popularity correlated with a linear increase in the other attributes however this is proven not to be the case due to the variance in the attributes from repository to repository. This has identified that every repository is unique due to factors not encompassed by this study by transient qualities such as activity rate of the contributors and its purpose, an extremely popular project is not entirely one that has a lot of commits or lines of code, possibly it is popular because it fits a niche. The study was to an extent build on the assumption that as popularity increases the more the other attributes will also correlate, which is a flawed assumption.

The multi-language aspect of the study is also quite significant, other research typically has a reduced scope such as tackling one particular use case or programming language. This study tries to present a general approach to apply machine learning techniques to classify popularity, which not successful was a novel idea and has significance in the fact it seems to not be a possibility, which fills a hole in the research (to my knowledge).

**5. Threats to validity**

It is crucial to considered aspects of the study which could skew the results, initially the dataset size should be evaluated. Currently generating the is a manual process of polling the API and visiting the repository web page which is a time consuming process, this restricts the size of the dataset. In essence this restricts the reliability of the results as they sample size is quite small but in this scenario it is acceptable.

Is a multi-language study for such a small dataset sensible? It could be more prudent to focus on a single language which would take greater advantage of the limited instances more fully for a more complete analysis. This is further confounded when its considered that some languages (HTML) will have greatly varying data over others (for example C) in the case of some metrics such as LOC which will of course in a typical C project eclipse the HTML LOC. However despite these the goal was to provide a novel study and to my knowledge this stance has not yet been taken using these data mining techniques as a form of comparison which adds significance and makes the decision worthwhile, partially I am handicapped by the scope of the assignment, in a fill fledged paper more samples for each language could be introduced.

External validity should be considered, that is the extent the findings can be generalized. In this particular study this form of validity is quite weak, again as a product of the small dataset. As mentioned before Naive Bayes has an assumption about the independence of the data, it is impossible at this stage to verify that in this dataset however logic suggests that repository data such as commits will have an impact on the popularity so it is a reasonable threat to bring up.

**6. Future Work**

I believe that to ensure a robust report it is important to consider avenues of research to consider in future to improve what has been performed in this study. To combat the small dataset issue it would be wise to design a workbench (which possibly extends of Darwin) to enable the collection of the dataset automatically by manipulating HTTP requests using Ajax callback functionality within JQuery. Once the larger dataset is formed is becomes possible to generalize the findings across a larger domain and enable complete evaluation of the results and possibly find novel contributions to the field. Overall a similar approach leveraging an expanded scope could provide a research worthy area of investigation.

**References**

* 1. Language Trends on GitHub, https://github.com/blog/2047-language-trends-on-github, August 19, 2015
  2. <http://www.d.umn.edu/~padhy005/Chapter5.html>
  3. The lecture Notes (on classification)
  4. S.B. Kotsiantis, Supervised Machine Learning: A Review of Classification Techniques, Informatica 31(2007) 249-268, 2007
  5. Landis, J.R.; Koch, G.G. (1977). "The measurement of observer agreement for categorical data". Biometrics 33 (1): 159–174. doi:10.2307/2529310. JSTOR 2529310. PMID 843571.
  6. Jyoti Sheoran et al, Understanding "watchers" on GitHub, MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, Pages 336-339
  7. Jesus M. Gonzalez-Barahona et al, Studying the laws of software evolution in a long-lived FLOSS Project, JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS J. Softw. Evol. and Proc. 2014; 26:589–612
  8. Xu Ben, Shen Beijun,Yang Weicheng, Mining Developer Contribution in Open Source Software Using Visualization Techniques, Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference, 16-18 Jan. 2013, 934 – 937

**Appendix**

A)

@RELATION 'repositories'

@ATTRIBUTE 'Language' {JavaScript, Java, Ruby, PHP, Python, CSS, C++, C#, C, HTML}

@ATTRIBUTE 'Commits' NUMERIC

@ATTRIBUTE 'Branches' NUMERIC

@ATTRIBUTE 'Watchers' NUMERIC

@ATTRIBUTE 'Forks' NUMERIC

@ATTRIBUTE 'Issues' NUMERIC

@ATTRIBUTE 'Releases' NUMERIC

@ATTRIBUTE 'Readme' {Yes, No}

@ATTRIBUTE '<100Contributors' {Yes, No}

@ATTRIBUTE 'LOC' NUMERIC

@ATTRIBUTE 'CommitInTheLastMonth' {Yes, No}

@ATTRIBUTE 'Pull Requests' NUMERIC

@ATTRIBUTE 'stars' {<500, 500-2000, 2000-5000, 5000-10000, >10000}

@data

C,1298,15,97,268,72,25,Yes,Yes,65591,Yes,7,<500 %https://github.com/edenhill/librdkafka

C,24,1,14,53,4,0,Yes,Yes,2389,No,0,<500 %https://github.com/karthick18/inception

C,81,1,107,132,0,0,Yes,Yes,1046,No,0,<500 %https://github.com/x0r1/jellyfish

C,378,127,397,25,15,5,Yes,Yes,6046,No,5,500-2000 %https://github.com/pedrovgs/DraggablePanel

C,104,2,132,179,52,7,Yes,Yes,9627,No,0,500-2000 %https://github.com/google/ios-webkit-debug-proxy

C,677,22,427,1209,58,8,Yes,Yes,15976,No,5,2000-5000 %https://github.com/square/dagger

C,11304,7,426,2355,387,0,Yes,No,321560,Yes,9,2000-5000 %https://github.com/codecombat/codecombat

C,266,7,600,3400,66,8,Yes,Yes,3927,No,43,5000-10000 %https://github.com/scottjehl/Respond

C,3,3,314,81415,657,0,Yes,Yes,46,No,5000,5000-10000 %https://github.com/octocat/Spoon-Knife

C,431,3,461,1785,22,42,Yes,Yes,4181,No,5,5000-10000 %https://github.com/desandro/masonry

C,4017,2,3621,12679,7,0,Yes,No,5730,Yes,13,>10000 %https://github.com/vhf/free-programming-books

C,3521,35,2355,13228,135,192,Yes,No,51675,Yes,66,>10000 %https://github.com/mbostock/d3

JavaScript,451,3,37,37,17,22,Yes,Yes,8946,No,1,<500 %https://github.com/conveyal/transitive.js

JavaScript,122,2,20,54,12,2,Yes,Yes,3446,No,3,<500 %https://github.com/ded/morpheus

JavaScript,95,3,52,99,6,13,Yes,Yes,1694,No,2,500-2000 %https://github.com/stutrek/scrollMonitor

JavaScript,155,3,114,257,14,0,Yes,Yes,101582,No,1,500-2000 %https://github.com/MicrosoftEdge/static-code-scan

JavaScript,175,9,44,50,15,0,Yes,Yes,15080,No,0,500-2000 %https://github.com/begriffs/css-ratiocinator

JavaScript,2565,9,325,1748,252,13,Yes,No,21811,No,26,2000-5000 %https://github.com/novus/nvd3

JavaScript,845,2,342,1258,328,17,Yes,Yes,41839,No,123,2000-5000 %https://github.com/flot/flot

JavaScript,563,4,291,939,467,15,Yes,Yes,261134,No,30,2000-5000 %https://github.com/vitalets/x-editable

JavaScript,266,7,600,3438,66,8,Yes,Yes,3927,No,43,5000-10000 %https://github.com/scottjehl/Respond

JavaScript,431,3,461,1785,22,42,Yes,Yes,4181,No,5,5000-10000 %https://github.com/desandro/masonry

JavaScript,6545,15,3002,6879,449,44,Yes,No,134373,Yes,30,>10000 %https://github.com/facebook/react

JavaScript,6078,4,3251,10436,57,138,Yes,No,62959,Yes,30,>10000 %https://github.com/jquery/jquery

Java,64,1,13,28,14,2,Yes,Yes,3726,No,1,<500 %https://github.com/nzakas/cssembed

Java,51,3,17,71,1,0,Yes,Yes,2442,No,1,<500 %https://github.com/JorgeCastilloPrz/ExpandablePanel

Java,185,7,524,1098,31,21,Yes,Yes,234441,Yes,3,500-2000 %https://github.com/alibaba/jstorm

Java,509,2,135,485,45,23,Yes,Yes,30199,Yes,1,500-2000 %https://github.com/code-troopers/android-betterpickers

Java,677,22,427,1209,58,8,Yes,Yes,15979,No,5,2000-5000 %https://github.com/square/dagger

Java,31202,639,354,534,0,5212,Yes,No,1498133,Yes,20,2000-5000 %https://github.com/JetBrains/kotlin

Java,73,5,270,847,3,0,Yes,Yes,3269,Yes,3,2000-5000 %https://github.com/lgvalle/Material-Animations

Java,891,58,825,2629,138,20,Yes,Yes,15431,Yes,28,5000-10000 %https://github.com/square/picasso

Java,203,1,1105,1789,25,1,Yes,Yes,5352,Yes,3,5000-10000 %https://github.com/futurice/android-best-practices

Java,405,5,841,2579,62,5,Yes,Yes,9834,Yes,3,5000-10000 %https://github.com/greenrobot/EventBus

Java,21475,53,1622,5417,1008,148,Yes,No,1121692,Yes,3,>10000 https://github.com/elastic/elasticsearch

Java,4670,4,1125,2117,101,138,Yes,No,131796,Yes,3,>10000 %https://github.com/ReactiveX/RxJava

Ruby,342,5,34,296,34,22,Yes,No,11279,No,3,<500 %https://github.com/cschiewek/devise\_ldap\_authenticatable

Ruby,161,1,21,107,14,9,Yes,Yes,3453,No,3,<500 %https://github.com/maxjustus/sinatra-authentication

Ruby,1086,7,67,153,12,92,Yes,Yes,6753,No,5,500-2000 %https://github.com/wvanbergen/request-log-analyzer

Ruby,4670,4,1125,2117,101,138,Yes,No,8409,Yes,3,500-2000 %https://github.com/intridea/hashie

Ruby,6587,71,250,1072,33,71,Yes,No,11232,Yes,3,2000-5000 %https://github.com/cucumber/cucumber-ruby

Ruby,246,3,200,286,12,9,Yes,Yes,1810,No,4,2000-5000 %https://github.com/rails-api/rails-api

Ruby,43056,18,1022,2794,123,309,Yes,Yes,13564,Yes,3,5000-10000 %https://github.com/ruby/ruby

Ruby,857,16,414,3119,180,1,Yes,No,3222,No,98,5000-10000 %https://github.com/imathis/octopress

Ruby,49212,2,217,3517,63,114,Yes,No,223442,Yes,15,5000-10000 %https://github.com/caskroom/homebrew-cask

Ruby,57373,22,2214,12487,418,290,Yes,No,394621,Yes,508,>10000 %https://github.com/rails/rails

Ruby,7126,25,1269,5346,138,97,Yes,No,234165,Yes,3,>10000 %https://github.com/jekyll/jekyll

Ruby,18715,15,817,4767,0,159,Yes,No,88721,Yes,22,>10000 %https://github.com/discourse/discourse

PHP,448,2,33,47,2,33,Yes,Yes,?,No,1342,<500 %https://github.com/peteboere/css-crush

PHP,338,3,75,67,4,0,Yes,Yes,563,No,2,<500 %https://github.com/FriendsOfPHP/security-advisories

PHP,1344,3,90,175,120,37,Yes,Yes,5642,No,10,<500 %https://github.com/rocketeers/rocketeer

PHP,25343,5,324,2143,267,133,Yes,No,1018308,No,332,500-2000 %https://github.com/joomla/joomla-cms

PHP,586,5,101,381,84,70,Yes,Yes,7362,Yes,1,500-2000 %https://github.com/Maatwebsite/Laravel-Excel

PHP,40,1,287,150,1,0,Yes,Yes,2789,No,1,500-2000 %https://github.com/phptodayorg/php-must-watch

PHP,1002,11,244,450,128,2,Yes,Yes,44321,No,6,2000-5000 %https://github.com/twostairs/paperwork

PHP,1490,2,278,851,35,32,Yes,No,20410,No,10,2000-5000 %https://github.com/Seldaek/monolog

PHP,601,4,920,1930,4,0,Yes,Yes,8911,Yes,4,5000-10000 %https://github.com/domnikl/DesignPatternsPHP

PHP,1625,5,432,1252,14,6,Yes,No,13,Yes,33,5000-10000 %https://github.com/fzaninotto/Faker

PHP,4804,5,3097,7517,0,63,Yes,No,45,Yes,0,>10000 %https://github.com/laravel/laravel

PHP,25926,11,1068,4788,572,164,Yes,No,123,Yes,155,>10000 %https://github.com/symfony/symfony

Python,229,2,48,97,38,33,Yes,Yes,73054,No,0,<500 %https://github.com/sametmax/0bin

Python,2133,17,45,53,1,1,Yes,Yes,9002,No,0,<500 %https://github.com/guardian/alerta

Python,998,2,168,539,79,1,Yes,Yes,2114,No,23,500-2000 %https://github.com/Lasagne/Lasagne

Python,812,7,179,216,48,32,Yes,Yes,8321,No,48,500-2000 %https://github.com/spinnaker/spinnaker

Python,338,3,18,63,182,25,Yes,Yes,17343,No,0,500-2000 %https://github.com/kennethreitz/clint

Python,448,2,611,1596,163,23,Yes,Yes,20011,No,12,2000-5000 %https://github.com/fxsjy/jieba

Python,2207,3,293,364,53,45,Yes,Yes,57176,No,1,2000-5000 %https://github.com/nicolargo/glances

Python,1970,3,242,373,84,47,Yes,No,16321,No,31,2000-5000 %https://github.com/nate-parrott/Flashlight

Python,9,1,19,29,0,0,Yes,Yes,?,No,1837,<500 %https://github.com/mschwager/dhcpwn

Python,138,1,86,250,19,9,Yes,Yes,8332,No,48,500-2000 %https://github.com/JakeWharton/pidcat

Python,9493,31,287,1573,349,158,Yes,No,22333,Yes,31,2000-5000 %https://github.com/celery/celery

Python,261,7,293,364,19,5,Yes,Yes,10964,No,1,2000-5000 %https://github.com/facebook/chisel

CSS,84,2,40,88,0,2,Yes,Yes,343,No,0,<500 %https://github.com/kogakure/gitweb-theme

CSS,3,1,31,91,0,0,Yes,Yes,1231,No,0,<500 %https://github.com/codrops/PageLoadingEffects

CSS,81,1,236,145,1,0,Yes,Yes,765,No,0,500-2000 %https://github.com/AllThingsSmitty/must-watch-css

CSS,745,3,236,638,11,0,Yes,Yes,896,No,0,500-2000 %https://github.com/1sters/material\_design\_zh

CSS,233,1,60,190,32,10,Yes,Yes,123,No,0,500-2000 %https://github.com/rstacruz/flatdoc

CSS,1178,4,62,209,9,3,Yes,Yes,8641,No,2,500-2000 %https://github.com/mdo/github-buttons

CSS,30,1,80,49,7,0,Yes,Yes,78232,No,0,<500 %https://github.com/m242/maildrop

CSS,306,2,47,55,7,18,Yes,Yes,4832,No,0,<500 %https://github.com/HubSpot/tooltip

CSS,47,2,20,32,4,2,Yes,Yes,22453,No,2,<500 %https://github.com/bjork24/Unison

CSS,178,4,62,209,9,3,Yes,Yes,2890,No,2,500-2000 %https://github.com/mdo/github-buttons

CSS,210,2,48,180,6,31,Yes,Yes,2012,No,1,500-2000 %https://github.com/wavded/humane-js

CSS,210,2,48,180,6,31,Yes,Yes,6652,No,1,500-2000 %https://github.com/poole/poole

C++,144,4,22,78,12,24,Yes,Yes,33818,No,3,<500 %https://github.com/node-inspector/v8-profiler

C++,1540,1,29,82,47,0,Yes,Yes,28901,No,1,<500 %https://github.com/etexteditor/e

C++,209,1,71,86,0,207,Yes,Yes,106090,No,0,<500 %https://github.com/AutoHotkey/AutoHotkey

C++,135,2,156,148,6,2,Yes,Yes,248112,Yes,0,500-2000 %https://github.com/electronicarts/EASTL

C++,128,5,189,548,33,0,Yes,Yes,8921,No,9,500-2000 %https://github.com/codebutler/firesheep

C++,2826,2,213,391,96,0,Yes,Yes,183712,No,9,2000-5000 %https://github.com/paulasmuth/fnordmetric

C++,39074,13,497,3461,221,110,Yes,Yes,99221,Yes,221,2000-5000 %https://github.com/xbmc/xbmc

C++,2052,1,318,444,113,7,Yes,Yes,68998,Yes,19,2000-5000 %https://github.com/SFTtech/openage

C++,3692,7,1332,5709,396,11,Yes,No,144659,Yes,211,5000-10000 %https://github.com/BVLC/caffe

C++,18884,2,1105,7890,942,48,Yes,No,111234,Yes,58,5000-10000 %https://github.com/Itseez/opencv

C++,36373,17,1972,3127,1643,68,Yes,Yes,637281,Yes,15,>10000 %https://github.com/apple/swift

C#,488,2,49,233,41,0,Yes,Yes,831,No,23,<500 %https://github.com/migueldeicaza/MonoTouch.Dialog

C#,8,1,64,131,4,0,Yes,Yes,11022,No,5,<500 %https://github.com/wybory2014/Kalkulator1

C#,3348,15,162,270,312,32,Yes,Yes,146868,Yes,5,500-2000 %https://github.com/jaredpar/VsVim

C#,4029,9,249,885,58,50,Yes,Yes,293102,Yes,4,500-2000 %https://github.com/cefsharp/CefSharp

C#,13404,41,793,1248,2611,35,Yes,Yes,65212,Yes,107,2000-5000 %https://github.com/dotnet/roslyn

C#,106398,219,517,2171,0,316,Yes,Yes,29321,Yes,111,2000-5000 %https://github.com/mono/mono

C#,10041,10,1433,2274,938,3,Yes,Yes,2393113,Yes,39,5000-10000 %https://github.com/dotnet/corefx

C#,5212,3,978,710,0,1,Yes,Yes,647212,Yes,79,5000-10000 %https://github.com/dotnet/coreclr

C#,4520,92,667,1732,397,36,Yes,Yes,85391,No,18,5000-10000 %https://github.com/SignalR/SignalR

C#,175,3,53,90,39,10,Yes,Yes,4492,No,2,<500 %https://github.com/kohsuke/winsw

C#,1747,3,75,65,66,6,Yes,Yes,17789,Yes,0,<500%https://github.com/Pash-Project/Pash

C#,1612,3,131,227,238,26,Yes,Yes,5226,Yes,11,>10000%https://github.com/chocolatey/choco

HTML,66,1,38,80,20,4,Yes,Yes,7890,No,2,<500 %https://github.com/scotch-io/scotch-panels

HTML,15,1,44,108,0,0,Yes,Yes,14016,No,3,<500 %https://github.com/Aaaaaashu/Front-End-Style-Guide

HTML,586,2,57,283,35,0,Yes,Yes,88221,Yes,2,<500 %https://github.com/coursera-dl/edx-dl

HTML,373,13,121,641,120,19,Yes,Yes,9021,No,43,500-2000 %https://github.com/vitch/jScrollPane

HTML,248,3,94,94,0,15,Yes,Yes,67821,Yes,0,500-2000 %https://github.com/caiorss/Functional-Programming

HTML,203,2,75,98,19,13,Yes,Yes,13411,No,2,500-2000 %https://github.com/micha/resty

HTML,160,2,141,432,11,6,Yes,Yes,864,No,4,500-2000 %https://github.com/sofish/typo.css

HTML,482,4,144,443,60,27,Yes,Yes,7212,Yes,21,500-2000 %https://github.com/grangier/python-goose

HTML,26,1,49,78,3,8,Yes,Yes,?,No,2001,500-2000 %https://github.com/1000ch/grd

HTML,611,36,124,192,26,2,Yes,Yes,65120,No,17,<500 %https://github.com/strangeioc/strangeioc

HTML,338,2,28,358,11,9,Yes,Yes,890,Yes,2,<500 %https://github.com/Huxpro/huxpro.github.io

HTML,122,5,26,36,0,6,Yes,Yes,1181,No,0,<500 %https://github.com/bensmithett/style

B) Textual representation of the J48 chart

Watchers <= 189

| Watchers <= 124: <500 (49.0/16.0)

| Watchers > 124: 500-2000 (10.0/1.0)

Watchers > 189

| Watchers <= 793

| | Forks <= 1248: 2000-5000 (21.0/6.0)

| | Forks > 1248

| | | LOC <= 11279: 5000-10000 (7.0)

| | | LOC > 11279: 2000-5000 (10.0/3.0)

| Watchers > 793

| | <100Contributors = Yes

| | | Forks <= 2794: 5000-10000 (7.0)

| | | Forks > 2794: >10000 (2.0)

| | <100Contributors = No: >10000 (14.0/3.0)