Software & Data Engineering

Project Report

IIT Jodhpur

Project Title - A Machine Learning Approach to Improve the Detection of CI Skip Commits

Project Owners:

- Ravi Shankar Kumar(M21AIE247)
- Debonil Ghosh(M21AIE225)

Project Instructor :

- Sumit Karla

Github Repo :
https://github.com/debonil/skip-commit-detection

1. Team Member Details :

Member -1 :

Name - Ravi Shankar Kumar

Profession - IT & Softwares

Current Company - EPAM System India Pvt Ltd.

Title : Quality Architect -I

Educational Background - B.Tech in Computer Science from IIIT Kolkata & Pursuing M.Tech in AI & Machine learning from IIT Jodhpur

Total Experience - 15 years

Member -2 :

Name - Debonil Ghosh

Profession - IT & Softwares

Current Company - Centre for Railway Information Systems (CRIS)

Title - Senior Software Engineer.

Educational Background - B.Tech in Computer Science from Govt College of Engineering and Textile Technology, Serampore & Pursuing M.Tech in AI & Machine

learning from IIT Jodhpur

Total Experience - 8 years

2. Introduction

Today Continuous Integration(CI) - Continuous Delivery(CD) tools and frameworks are being used in almost every software project be it large or small, be it in healthcare domain or social media or any other domain. The biggest use case of CI tools is to automatically build, test & deploy codes on lower or higher environments with each commit being merged to master or main branches, sometimes even creation of pull requests triggers a build(the way it is configured) . But we know that since build pipelines are of multiple phases starting from environment preparation to cleaning to building the code & obviously running the tests which means they are bound to be a time consuming process. Since there are many developers working in a team and they don't just belong to core development team they can belong to IT infrastructure, DevOps teams, Ui\Ux teams who generally not work on core source code but some configuration or files which are non-source which means if they also go ahead and create a PR or if their PR is merged then it will trigger a build. Let’s leave these kinds of commits from members from these teams, let suppose a core developer just updated a small comment in code or she just updated a small change in README.md files the build is going to trigger. What we can observe here is that for every small or tiny change which might not affect the functionality of an application is going to trigger build and that is going to take time which means that it is a waste of time & money. What if automatically CI\CD tools can intelligently find out how worth is this commit to trigger the build and it only triggers the build for the commits for which make sense and not all. There are many works which has been done previously and one of such work which actually tries to solve this problem and that is Rule based approach where you defined set of rules and CI tools will skip that commit but limitation of such approach is that rules are endless there is not end to rules because there can be many file types, many use cases on different platform and many many other cases. Understanding this limit Rabe Abdalkareem did work on machine learning based approach where he has used Decision tree with 23 features to detect if a commit is CI-SKIP or not and he has achieved the average AUC equal to 0.89 and for cross project the average AUC achieved was 0.74. The work done by Rabe & team is first of its kind where machine learning is used and since it has a good AUC number and it actually solves

real problems in software industry we picked this as our project as we also feel that this paper has capability to solve a genuine real life problem which can save dollars to a great extent. The main objective of our project is to demonstrate the same or better results using the machine learning approach proposed in this paper.

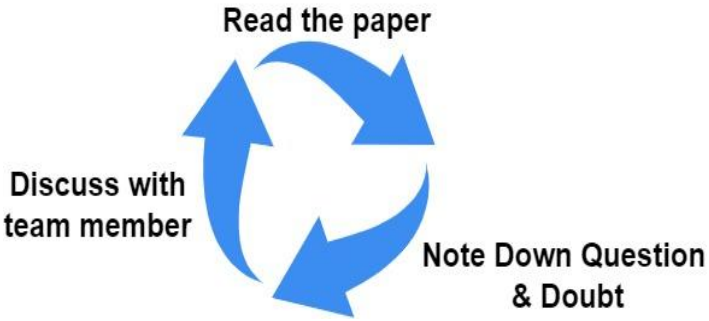
3. Summary of work done :

First thing first, we built a good understanding about the IEEE paper that we selected and referenced papers(papers which are cited as knowledge source in original paper) so that we can implement it and improve the result or at least demonstrate the same level of results. Secondly we went and searched for a project in github that we can use to train & test our modelson. We used BigQuery to scan through the Github projects to find the right project. The project which we picked must have a sufficient number of CI-Skipped commits so that we can get the labeled data and measure the results of our model. We decided on the features or probable features which we are going to use in our machine learning model.Now one of the biggest tasks was to get the data for training & testing, by data here we mean test data that can be used as a feature & their values for our machine learning model from the chosen project. Post gathering the data we processed the data so that it is ready to be consumed by the ML models. Once we had data for each feature in a format that can be used we hyper-parameter tuning with different machine learning models and their changeable parameters so that we can get the best model and best parameters values for the same so that we can get the best results. Post hyper parameter tuning we trained the model on the larger data and then tested the model on seen data. We validated the selected model on unseen data. For unseen data we used a different github project then the one which we had selected for training & testing. We compared the results for different models on training, test & validation sets. We demonstrated the results & published it.

4. Work Done

a. Knowledge building

It was very important for us to build the right & sufficient knowledge before we move on with the implementation of the project & the approach we use to do so is shown in diagram below.



(We did couple of iteration of tasks mentioned in above diagram)

b. Finding the right test data.

For any machine learning & artificial intelligence project to be successful, the key is not the right model to choose or right parameter to choose but the key is finding the right set of training & testing data. If you do not have the right data and not just right data, the data must be also in the right format to be successful. And to find the right test data we started by first analyzing the github projects mentioned in the research paper which we had picked. We tried to answer the question why only below projects were picked and not others, however authors of the paper have already mentioned the reason but there is always more to it then what we see in words. So we tried finding the facts and what came out to be is that the project must not have too many files which do not belong to the language of domain which was java for us here. Suppose if there are too many files of javascript in the project as compared to java the model will not be trained well. Project must not just have a sufficient amount of CI-Skip commits but also it must not be too many otherwise test data will be biased and this will result in lower performance. After understanding the properties of right test data and here for us it was the right github project. We started the search but again Github is a ocean of projects so finding the right project would have been impossible without the help of BigQuery. Paper has used these many github projects as testing as shown in the list below and we also decided to with the same list of project other then one which was not available as test data.

TABLE 3
Shows Projects in the Testing Dataset

Project	LOC	Classes	Methods	Time Period	# of Commits[§]	% of Skipped Commits
TracEE Context-Log	12,386	345	1,426	2014-12-12 - 2017-03-14	216	29.63
SAX	7,200	75	438	2015-06-20 - 2017-03-23	372	23.66
Trane.io Future	5,185	119	898	2017-02-05 - 2017-08-04	247	18.62
Solr-iso639-filter	2,335	82	299	2013-08-20 - 2015-06-16	408	41.42
jMotif-GI	4,689	54	336	2015-06-20 - 2017-03-29	345	12.17
GrammarViz	6,798	89	403	2014-08-22 - 2017-03-13	417	13.67
Parallec	12,452	242	1,112	2015-10-28 - 2017-07-09	129	56.59
CandyBar	13,702	206	1,025	2017-02-22 - 2017-09-26	242	69.01
SteVe	14,078	358	1,332	2015-10-07 - 2016-11-16	298	19.46
Mechanical Tsar	4,008	108	478	2015-05-06 - 2017-03-19	388	34.54
Average	**8,283.30**	**167.80**	**774.70**	-	**306.20**	**31.88**
Median	**6,999.00**	**113.50**	**688.00**	-	**321.50**	**26.65**

c. Finding the right set of features.

Another crucial thing for a machine learning project to be successful is do we have identified the right set of features that we are going to train our model on. Definitely we could see from the paper there were 23 features used as listed below that were used by this paper to train the model and it is already mentioned that out of 23 features the most important ones are no developers modified the files, commit messages & CI-Skip rules. We went ahead & picked up all the features from the paper and started analysis.We took all the features and started looking at the ***Github APIs*** and we created a feature list to track which one we will be able to extract and how we are going to extract.

Online copy of our feature list & analysis can be found here
https://tinyurl.com/mvnwvm4s

We extensively discussed each and every feature to answer the following basic questions like

- How important are they?
- How are we going to retrieve them?
- How will we convert them to machine learning consumable values?
- How are we going to clean the unwanted data?

In short we did feature engineering ourselves and not simply took all the features mentioned in the paper. And these are the features we finally selected and we noticed that even we selected bit lesser features then what was done in paper still our results are comparably same as in paper.

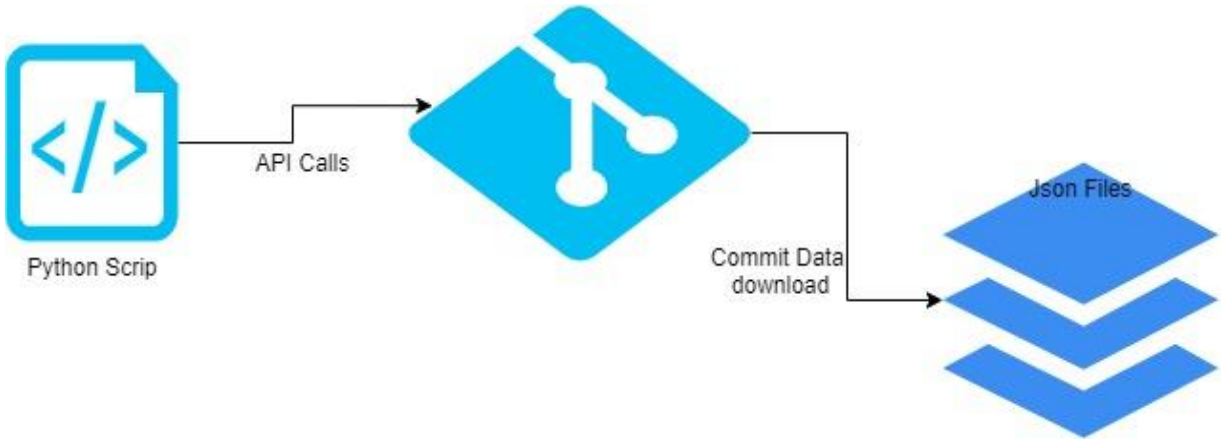
Dim.	Name	Definition (Inspired from)	Rational	Data Type	Retrieval?
Diffusion	**ND**	**Number of modified directories**	Build commits that are based on changing several directories are more likely not to be CI skipped.	**Integer**	Calculated
	NF	**Number of modified files**	Build commits that changes multiple files are more likely to modify source code files which should be built.	**Integer**	Direct
	EN	**Distribution of modified code across each file (i.e., Entropy)**	Changes with high entropy are more likely to have a large number of file changes that make the changes more complicated and have high chances to modify the source code.	**Float**	Calculated
Size	**LA**	**Lines of code added**	It is clear that the more lines of code added shows the need to build the system and run the test cases.	**Integer**	Direct
	LD	**Lines of code deleted**	The more lines of code deleted the more the need to run the continuous integration process.	**Integer**	Direct
Purpose	**FIX**	**Whether or not the change is a defect fix**	Fixing a defect means that more code is modified or added that need to be tested after the change through running the continuous integration process.	**Boolean**	Direct
	MR	**If the commit is a merge commit.**	The number of parents of a commit shows if the commit is a merge commit which required to running the continuous integration process to show that the merged code integrated integration process to show that the merged safely to the project code integrated	**Boolean**	Direct
History	**NDEV**	**The number of developers that changed the modified files ([17], [30], [32], [40], [67]).**	The larger the number of developers changed the modified files,the risker the changes are that require running the continuous integration process to make sure the changes do not break the integration process to make sure project or fail tests.the changes do not break the	**Number**	Calculated

Text	**CM**	**Terms appear in the commit messages. We weight the terms using tf-tdf after removing English stop words.**	Commit message are more likely to contain useful information about the type of changes in the commit (e.g., changes a readme files).	**String**	Calculated
CI-skip Rules	**DOC**	**If the commit changes non-source code files ([1]).**	Based on the devised rules if a commit changes mainly non-source code, it is likely to be CI skipped.	**Boolean**	Calculated
	MET	**If the commit modifies meta files such as git ignore ([1]).**	Based on the devised rules if a commit changes mainly meta files, it is likely to be CI skipped.	**Boolean**	Calculated
	COM	**If the commit modifies source code comments ([1]).**	From the devised rules if the changes in a commit are mainly related to source code comments, it is likely to be a CI skip commit.	**Boolean**	Calculated
	BLD	**If the commit modifies the version in the project ([1]).**	From the devised rules if the changes in a commit are mainly preparing for release or changing release version, it is likely to be a CI skip commit.	**Boolean**	Calculated

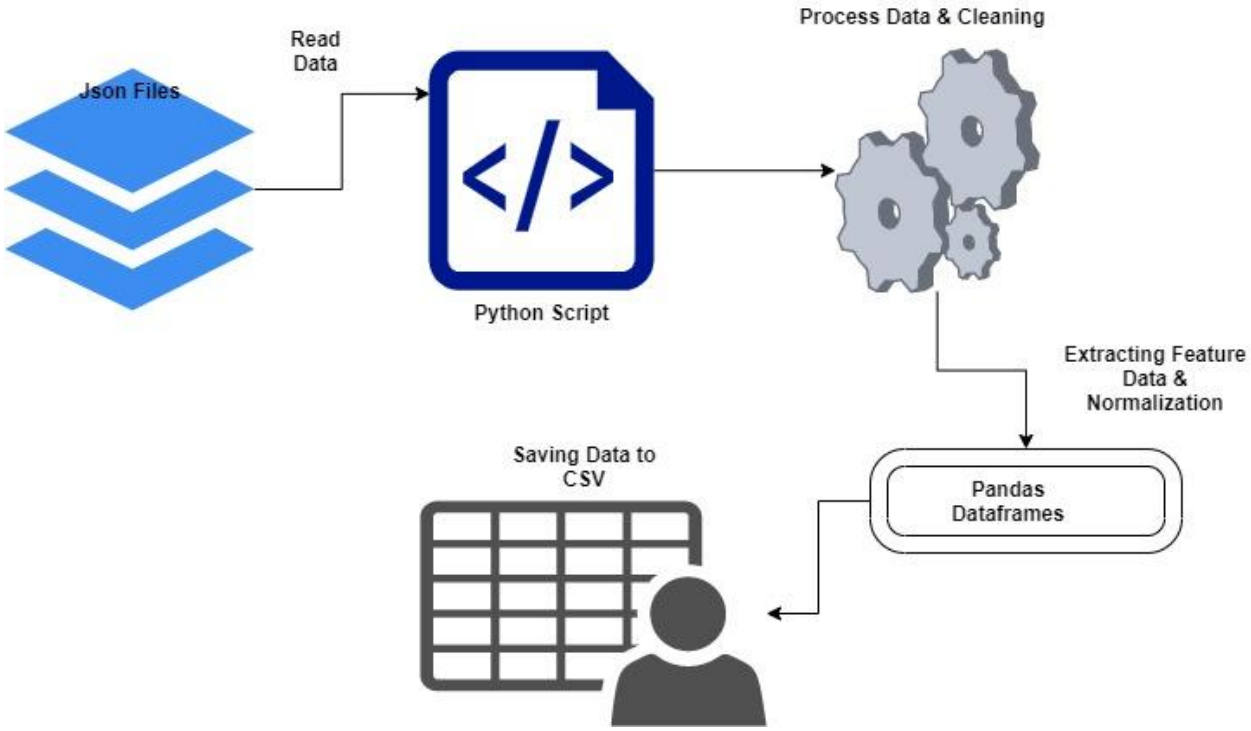
d. Extracting the data, cleaning & pre-processing.

To extract the data we heavily used the GitHub APIs. Mainly we used APIs related to commits obviously because we are working on CI-Skip and to do so we used python script to hit the APIs to read data using. We saved the data in Json file format for different repositories. Our script basically first downloads the commits data in json format then from the same json format we load the data into python objects and then these objects we use to process & extract the features post cleaning the junk data which is not needed.

Data Transfer from Git Remote to local



Cleaning & Processing of Data to make it ML model consumable.

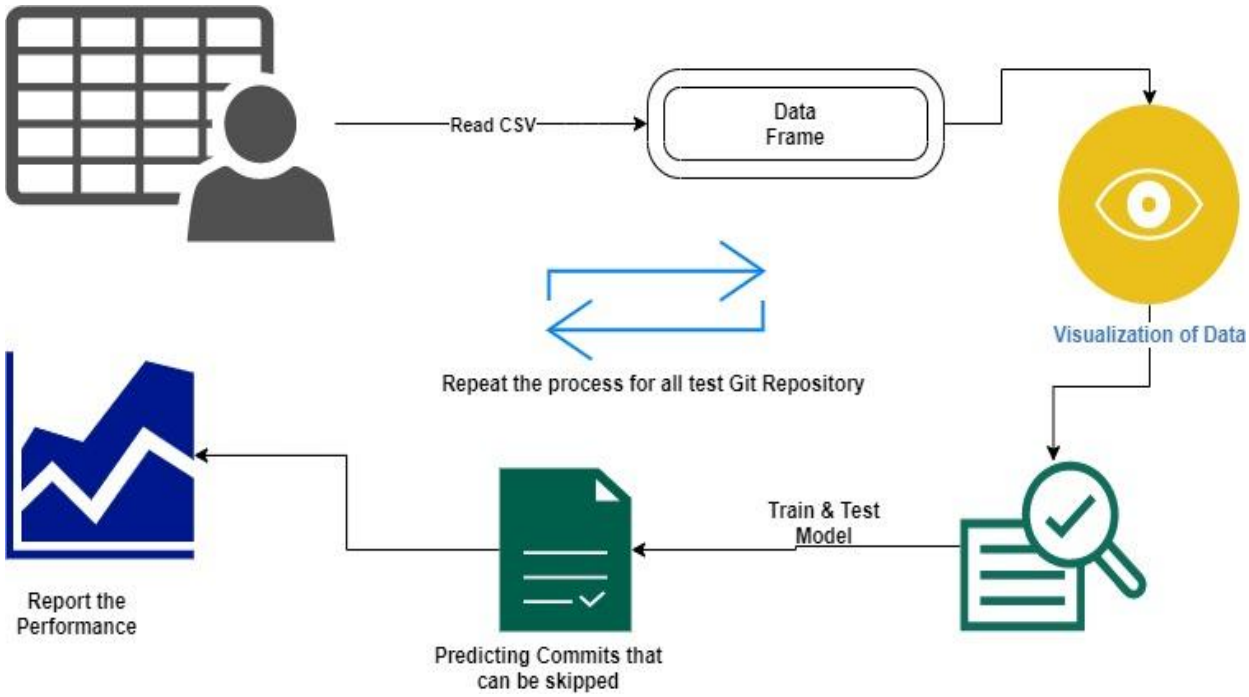


e. Model Training & Testing:

A training model is a dataset that is used to train an ML algorithm. It consists of the sample output data and the corresponding sets of input data that have an influence on the output. The training model is used to run the input data through the algorithm to correlate the processed output against the sample output. The result from this correlation is used to modify the model.

This iterative process is called “model fitting”. The accuracy of the training dataset or the validation dataset is critical for the precision of the model. Model training in machine language is the process of feeding an ML algorithm with data to help identify and learn good values for all attributes involved. There are several types of machine learning models, of which the most common ones are supervised and unsupervised learning.Supervised learning is possible when the training data contains both the input and output values. Each set of data that has the inputs and the expected output is called a supervisory signal. The training is done based on the deviation of the processed result from the documented result when the inputs are fed into the model.
Unsupervised learning involves determining patterns in the data. Additional data is then used to fit patterns or clusters. This is also an iterative process that improves the accuracy based on the correlation to the expected patterns or clusters. There is no reference output dataset in this method.

Our problem is a clear case of Supervised learning where CI-Skip commits are taken as labeled data sets to train the model and we have split the cleaned data into test & train data randomly and have trained the Models. Basically we took data from one repository which is nothing but the cleaned data present in CSVs and then we transform these data as data frames and feed machine learning models with the train & test data to make it learn.



f. Reporting the performance of Model:

These are the different performance metrics we are using to evaluate the model

- Accuracy
- Confusion Matrix
- Precision
- Recall
- F-Score
- AUC(Area Under the Curve)-ROC

I. Accuracy : The accuracy metric is one of the simplest Classification metrics to implement, and it can be determined as the number of correct predictions to the total number of predictions.
It can be formulated as:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

II. Confusion Matrix : A confusion matrix is a tabular representation of prediction outcomes of any binary classifier, which is used to describe the performance of the classification model on a set of test data when true values are known.
The confusion matrix is simple to implement, but the terminologies used in this matrix might be confusing for beginners.
A typical confusion matrix for a binary classifier looks like the below image(However, it can be extended to use for classifiers with more than two classes).

True Positive(TP): In this case, the prediction outcome is true, and it is true in reality, also.
True Negative(TN): in this case, the prediction outcome is false, and it is false in reality, also.
False Positive(FP): In this case, prediction outcomes are true, but they are false in actuality.
False Negative(FN): In this case, predictions are false, and they are true in actuality.

III. Precision : The precision metric is used to overcome the limitation of Accuracy. The precision determines the proportion of positive predictions that was actually correct. It can be calculated as the True Positive or predictions that are actually true to the total positive predictions (True Positive and False Positive).

$$Precision = \frac{TP}{(TP + FP)}$$

IV. Recall or Sensitivity : It is also similar to the Precision metric; however, it aims to calculate the proportion of actual positives that was identified incorrectly. It can be calculated as True Positive or predictions that are actually true to the total number of positives, either correctly predicted as positive or incorrectly predicted as negative (true Positive and false negative).

The formula for calculating Recall is given below:

$$\text{Recall} = \frac{TP}{TP+FN}$$

V. F-Scores : F-score or F1 Score is a metric to evaluate a binary classification model on the basis of predictions that are made for the positive class. It is calculated with the help of Precision and Recall. It is a type of single score that represents both Precision and Recall. So, the F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.
The formula for calculating the F1 score is given below:

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$

VI. AUC-ROC : Sometimes we need to visualize the performance of the classification model on charts; then, we can use the AUC-ROC curve. It is one of the popular and important metrics for evaluating the performance of the classification model.
Firstly, let's understand the ROC (Receiver Operating Characteristic curve) curve. ROC represents a graph to show the performance of a classification model at different threshold levels. The curve is plotted between two parameters, which are:
True Positive Rate
False Positive Rate
TPR or true Positive rate is a synonym for Recall, hence can be calculated as:

$$TPR = \frac{TP}{TP + FN}$$

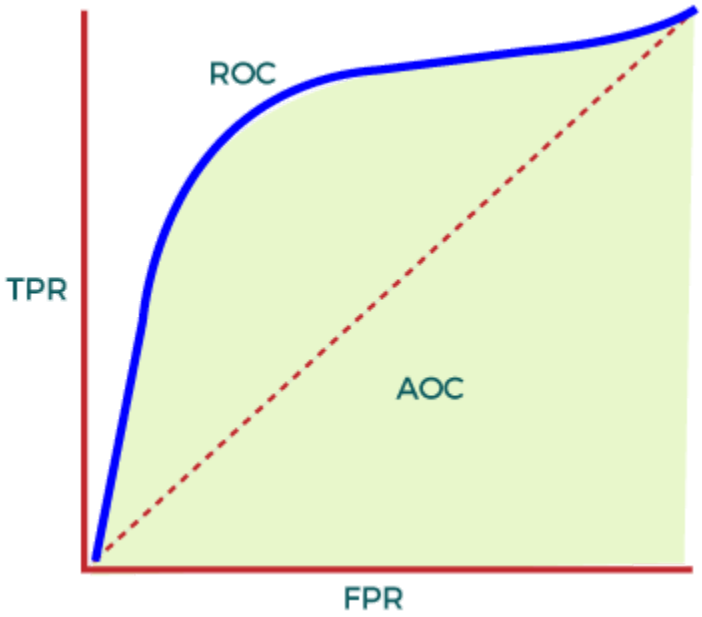
FPR or False Positive Rate can be calculated as:

$$TPR = \frac{FP}{FP + TN}$$

To calculate value at any point in a ROC curve, we can evaluate a logistic regression model multiple times with different classification thresholds, but this would not be much efficient. So, for this, one efficient method is used, which is known as AUC.

AUC: Area Under the ROC curve

AUC is known for Area Under the ROC curve. As its name suggests, AUC calculates the two-dimensional area under the entire ROC curve, as shown below image:



AUC calculates the performance across all the thresholds and provides an aggregate measure. The value of AUC ranges from 0 to 1. It means a model with 100% wrong prediction will have an AUC of 0.0, whereas models with 100% correct predictions will have an AUC of 1.0.

5. Results & Observations

Github Repository wise Model Performance Reporting

Dropwizard Github Repository- Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.973	0.499	0.000	0.499	0.000	0.000
Linear SVM	0.975	0.500	0.000	0.500	0.000	0.000
RBF SVM	0.975	0.500	0.000	0.500	0.000	0.000
Gaussian Process	0.974	0.500	0.000	0.500	0.000	0.000
Decision Tree	0.968	0.496	0.000	0.496	0.000	0.000
Random Forest	0.972	0.498	0.000	0.498	0.000	0.000
Neural Net	0.975	0.500	0.000	0.500	0.000	0.000
AdaBoost	0.975	0.500	0.000	0.500	0.000	0.000
Naive Bayes	0.975	0.500	0.000	0.500	0.000	0.000
QDA	0.975	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.975	0.500	0.000	0.500	0.000	0.000
Ridge Classifier	0.975	0.500	0.000	0.500	0.000	0.000

eBay parallec Github Repository - Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.973	0.976	0.970	0.976	0.941	1.000
Linear SVM	1.000	1.000	1.000	1.000	1.000	1.000
RBF SVM	1.000	1.000	1.000	1.000	1.000	1.000
Gaussian Process	0.973	0.976	0.970	0.976	0.941	1.000
Decision Tree	0.946	0.938	0.933	0.938	1.000	0.875
Random Forest	1.000	1.000	1.000	1.000	1.000	1.000
Neural Net	1.000	1.000	1.000	1.000	1.000	1.000
AdaBoost	0.946	0.952	0.941	0.952	0.889	1.000
Naive Bayes	0.568	0.500	0.000	0.500	0.000	0.000
QDA	0.568	0.500	0.000	0.500	0.000	0.000
Logistic Regression	1.000	1.000	1.000	1.000	1.000	1.000
Ridge Classifier	1.000	1.000	1.000	1.000	1.000	1.000

GrammarViz2 Github Repository - Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_s	precision_	recall_sco
Nearest Neighbors	0.813	0.581	0.222	0.581	0.176	0.300
Linear SVM	0.884	0.621	0.316	0.621	0.333	0.300
RBF SVM	0.884	0.621	0.316	0.621	0.333	0.300
Gaussian Process	0.884	0.621	0.316	0.621	0.333	0.300
Decision Tree	0.393	0.306	0.056	0.306	0.032	0.200
Random Forest	0.821	0.496	0.091	0.496	0.083	0.100
Neural Net	0.884	0.621	0.316	0.621	0.333	0.300
AdaBoost	0.821	0.812	0.444	0.812	0.308	0.800
Naive Bayes	0.911	0.500	0.000	0.500	0.000	0.000
QDA	0.911	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.875	0.616	0.300	0.616	0.300	0.300
Ridge Classifier	0.875	0.616	0.300	0.616	0.300	0.300

jMotif_GI Github Repository - Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.875	0.500	0.000	0.500	0.000	0.000
Linear SVM	0.938	0.893	0.769	0.893	0.714	0.833
RBF SVM	0.875	0.500	0.000	0.500	0.000	0.000
Gaussian Process	0.969	0.911	0.870	0.911	0.909	0.833
Decision Tree	0.792	0.452	0.000	0.452	0.000	0.000
Random Forest	0.875	0.500	0.000	0.500	0.000	0.000
Neural Net	0.969	0.911	0.870	0.911	0.909	0.833
AdaBoost	0.865	0.494	0.000	0.494	0.000	0.000
Naive Bayes	0.875	0.500	0.000	0.500	0.000	0.000
QDA	0.875	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.969	0.911	0.870	0.911	0.909	0.833
Ridge Classifier	0.865	0.494	0.000	0.494	0.000	0.000

jMotif_SAX Github Repository - Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.958	0.841	0.811	0.841	1.000	0.682
Linear SVM	0.952	0.837	0.789	0.837	0.938	0.682
RBF SVM	0.952	0.837	0.789	0.837	0.938	0.682
Gaussian Process	0.958	0.841	0.811	0.841	1.000	0.682
Decision Tree	0.958	0.841	0.811	0.841	1.000	0.682
Random Forest	0.958	0.841	0.811	0.841	1.000	0.682
Neural Net	0.952	0.837	0.789	0.837	0.938	0.682
AdaBoost	0.869	0.500	0.000	0.500	0.000	0.000
Naive Bayes	0.869	0.500	0.000	0.500	0.000	0.000
QDA	0.869	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.952	0.837	0.789	0.837	0.938	0.682
Ridge Classifier	0.952	0.837	0.789	0.837	0.938	0.682

Ksclarke_solr-iso639 Github Repository - Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.774	0.747	0.684	0.747	0.788	0.605
Linear SVM	0.821	0.786	0.732	0.786	0.929	0.605
RBF SVM	0.821	0.779	0.716	0.779	1.000	0.558
Gaussian Process	0.830	0.794	0.743	0.794	0.963	0.605
Decision Tree	0.811	0.771	0.706	0.771	0.960	0.558
Random Forest	0.821	0.783	0.725	0.783	0.962	0.581
Neural Net	0.840	0.802	0.754	0.802	1.000	0.605
AdaBoost	0.557	0.468	0.000	0.468	0.000	0.000
Naive Bayes	0.594	0.500	0.000	0.500	0.000	0.000
QDA	0.594	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.802	0.774	0.720	0.774	0.844	0.628
Ridge Classifier	0.802	0.774	0.720	0.774	0.844	0.628

mtsar_ Github Repository - Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.670	0.601	0.441	0.601	0.464	0.419
Linear SVM	0.640	0.473	0.053	0.473	0.143	0.032
RBF SVM	0.690	0.562	0.311	0.562	0.500	0.226
Gaussian Process	0.670	0.548	0.298	0.548	0.438	0.226
Decision Tree	0.660	0.558	0.346	0.558	0.429	0.290
Random Forest	0.690	0.527	0.162	0.527	0.500	0.097
Neural Net	0.640	0.526	0.280	0.526	0.368	0.226
AdaBoost	0.610	0.451	0.049	0.451	0.100	0.032
Naive Bayes	0.300	0.475	0.453	0.475	0.299	0.935
QDA	0.670	0.486	0.000	0.486	0.000	0.000
Logistic Regression	0.670	0.539	0.267	0.539	0.429	0.194
Ridge Classifier	0.650	0.533	0.286	0.533	0.389	0.226

steve-community Repository - Model performance

classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.944	0.496	0.000	0.496	0.000	0.000
Linear SVM	0.951	0.500	0.000	0.500	0.000	0.000
RBF SVM	0.951	0.500	0.000	0.500	0.000	0.000
Gaussian Process	0.949	0.499	0.000	0.499	0.000	0.000
Decision Tree	0.912	0.479	0.000	0.479	0.000	0.000
Random Forest	0.944	0.496	0.000	0.496	0.000	0.000
Neural Net	0.944	0.496	0.000	0.496	0.000	0.000
AdaBoost	0.951	0.500	0.000	0.500	0.000	0.000
Naive Bayes	0.951	0.500	0.000	0.500	0.000	0.000
QDA	0.951	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.951	0.500	0.000	0.500	0.000	0.000
Ridge Classifier	0.951	0.500	0.000	0.500	0.000	0.000

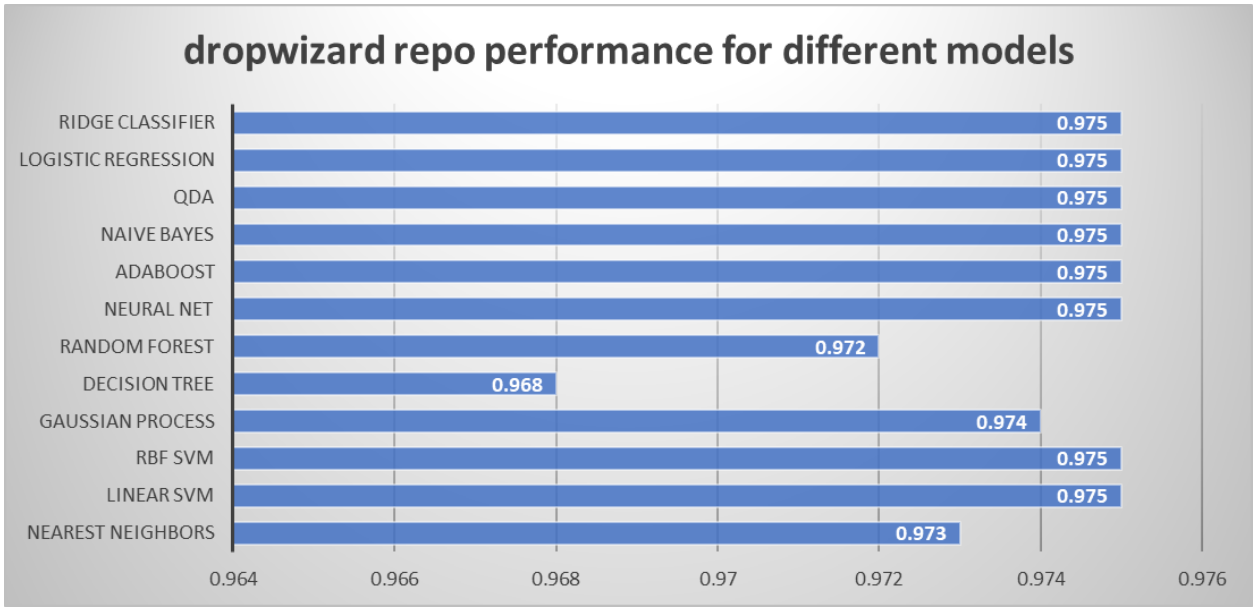
tracee_contextlogger Repository - Model performance

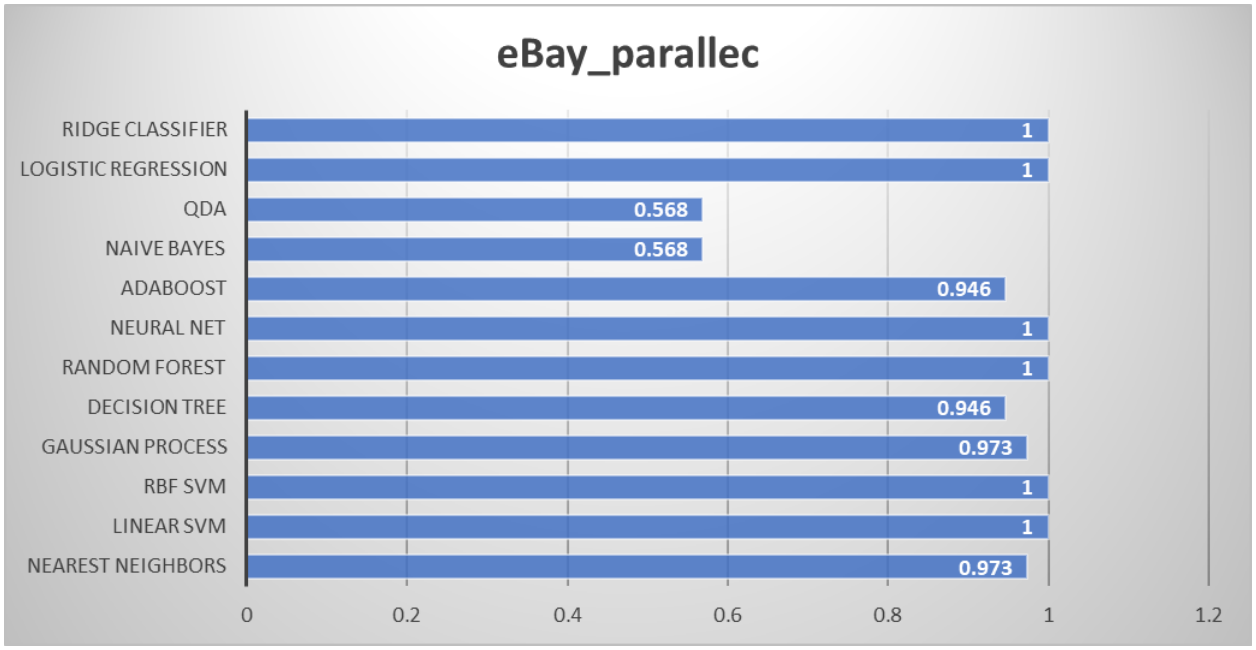
classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.957	0.943	0.857	0.943	0.800	0.923
Linear SVM	0.925	0.924	0.774	0.924	0.667	0.923
RBF SVM	0.957	0.943	0.857	0.943	0.800	0.923
Gaussian Process	0.935	0.930	0.800	0.930	0.706	0.923
Decision Tree	0.925	0.924	0.774	0.924	0.667	0.923
Random Forest	0.935	0.930	0.800	0.930	0.706	0.923
Neural Net	0.935	0.930	0.800	0.930	0.706	0.923
AdaBoost	0.871	0.538	0.143	0.538	1.000	0.077
Naive Bayes	0.860	0.500	0.000	0.500	0.000	0.000
QDA	0.860	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.935	0.930	0.800	0.930	0.706	0.923
Ridge Classifier	0.925	0.924	0.774	0.924	0.667	0.923

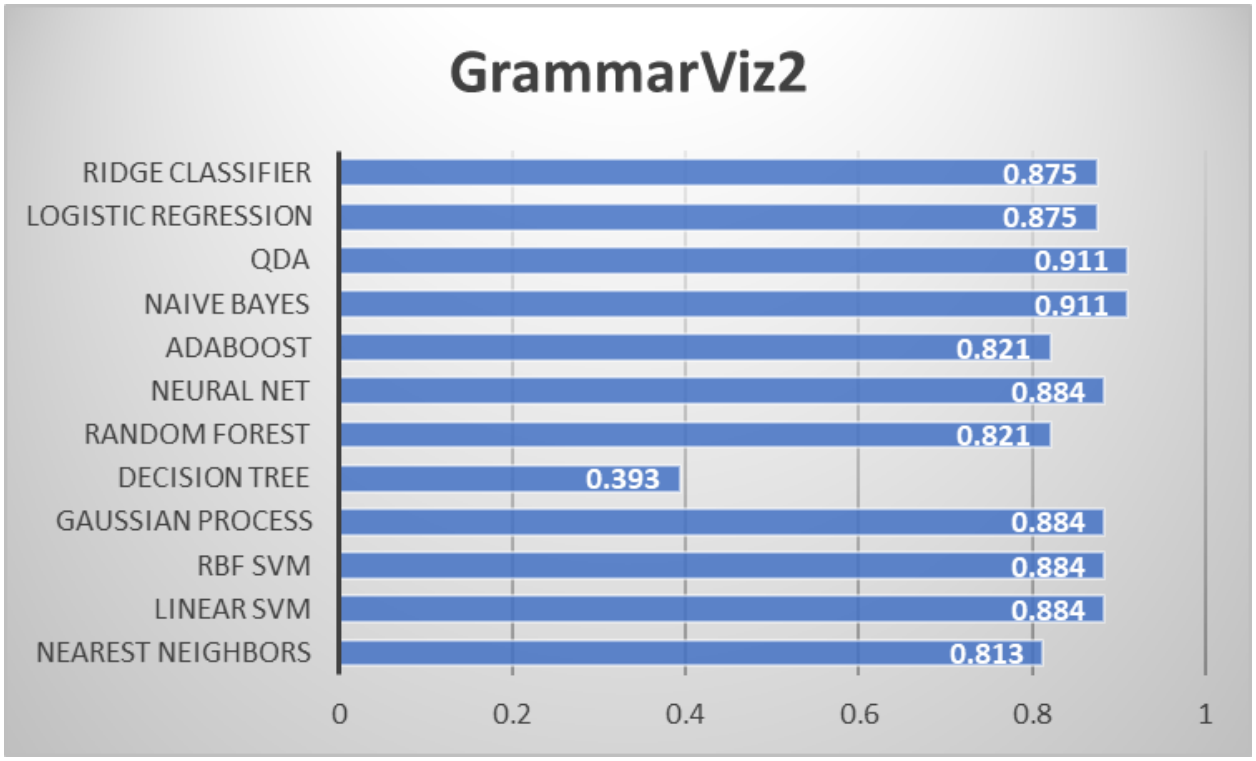
zixpo_candybar Repository - Model performance

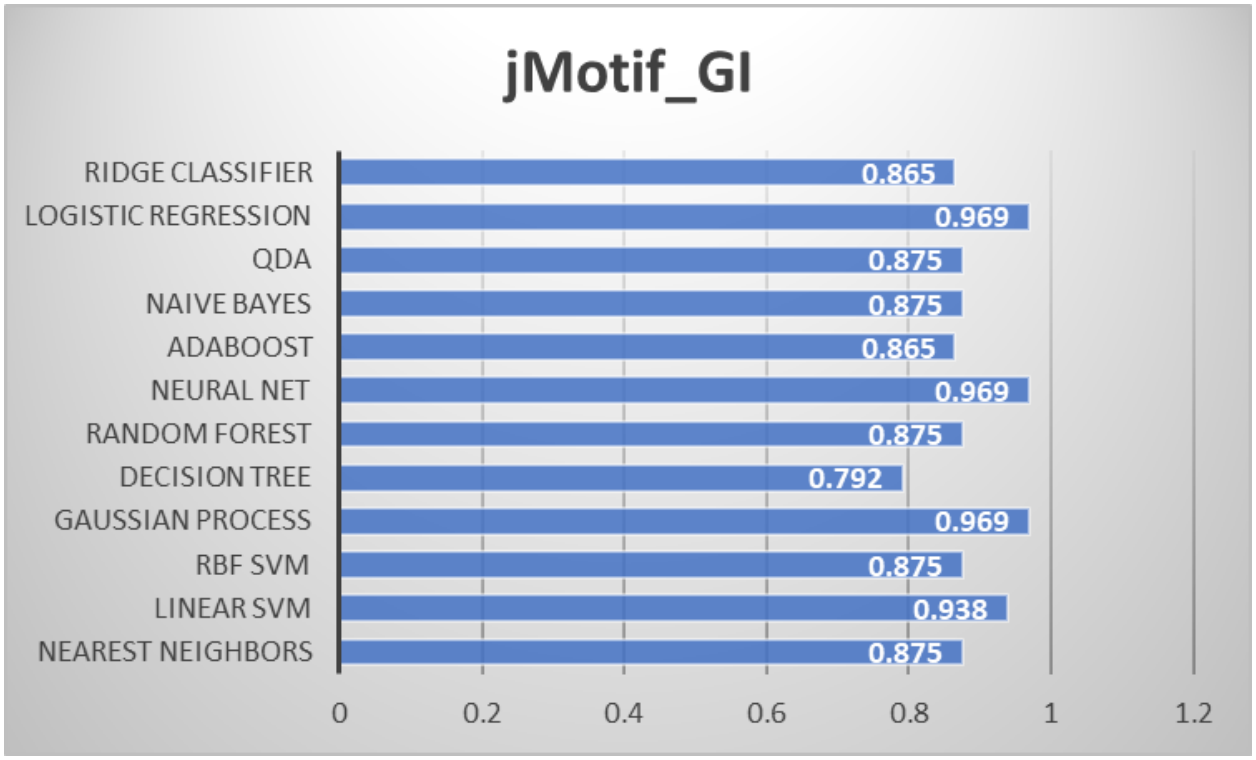
classifier	accuracy_score	balanced_accuracy_score	f1_score	roc_auc_score	precision_score	recall_score
Nearest Neighbors	0.929	0.496	0.000	0.496	0.000	0.000
Linear SVM	0.938	0.500	0.000	0.500	0.000	0.000
RBF SVM	0.938	0.500	0.000	0.500	0.000	0.000
Gaussian Process	0.925	0.524	0.100	0.524	0.200	0.067
Decision Tree	0.892	0.507	0.071	0.507	0.077	0.067
Random Forest	0.938	0.500	0.000	0.500	0.000	0.000
Neural Net	0.938	0.531	0.118	0.531	0.500	0.067
AdaBoost	0.925	0.493	0.000	0.493	0.000	0.000
Naive Bayes	0.938	0.500	0.000	0.500	0.000	0.000
QDA	0.938	0.500	0.000	0.500	0.000	0.000
Logistic Regression	0.933	0.529	0.111	0.529	0.333	0.067
Ridge Classifier	0.938	0.500	0.000	0.500	0.000	0.000

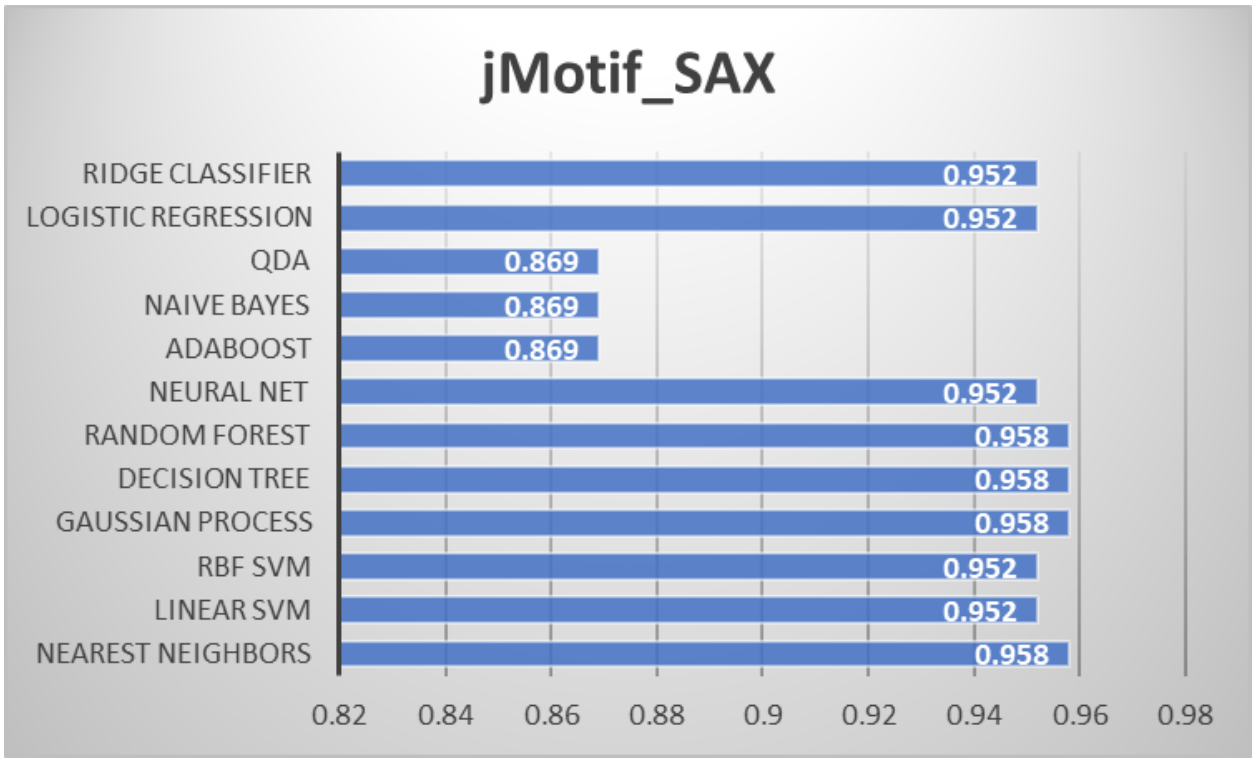
Github Repository wise Model Performance Trend

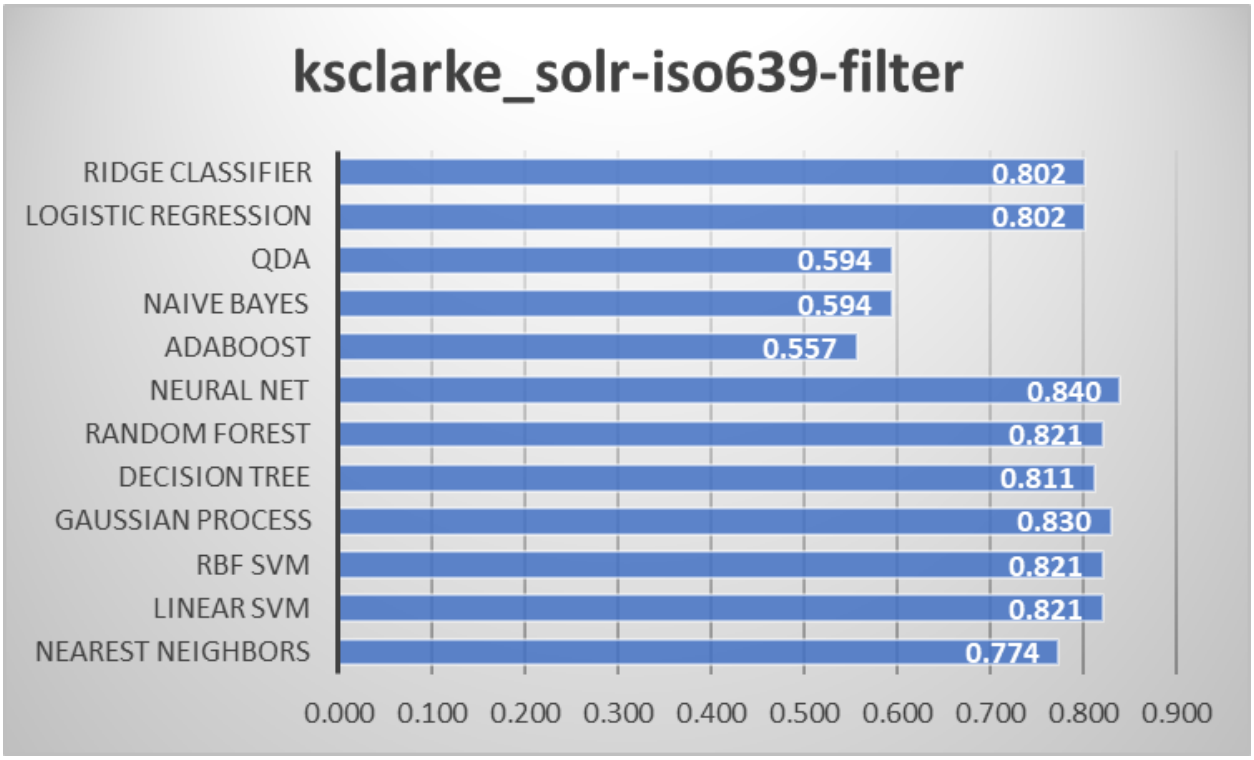


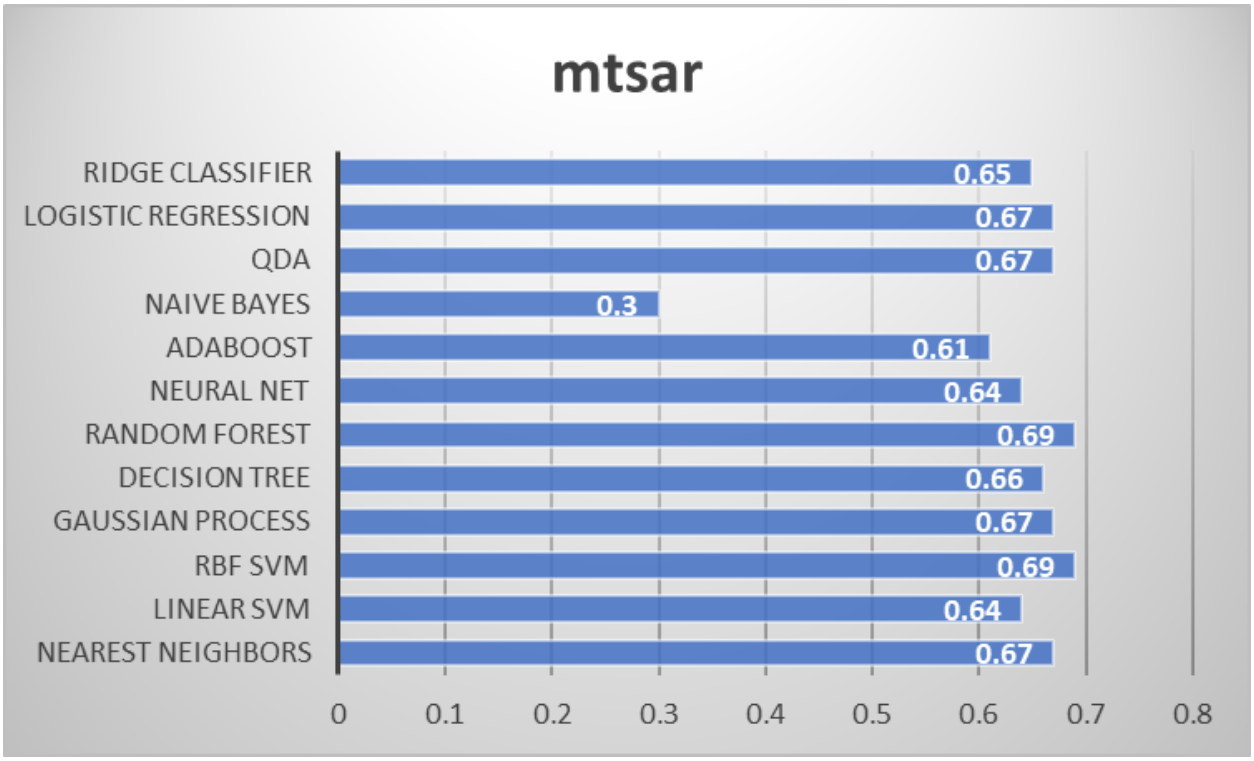

eBay_parallec
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
1
1
0.568
0.568
0.946
1
1
0.946
0.973
1
1
0.973
0
0.2
0.4
0.6
0.8
1
1.2

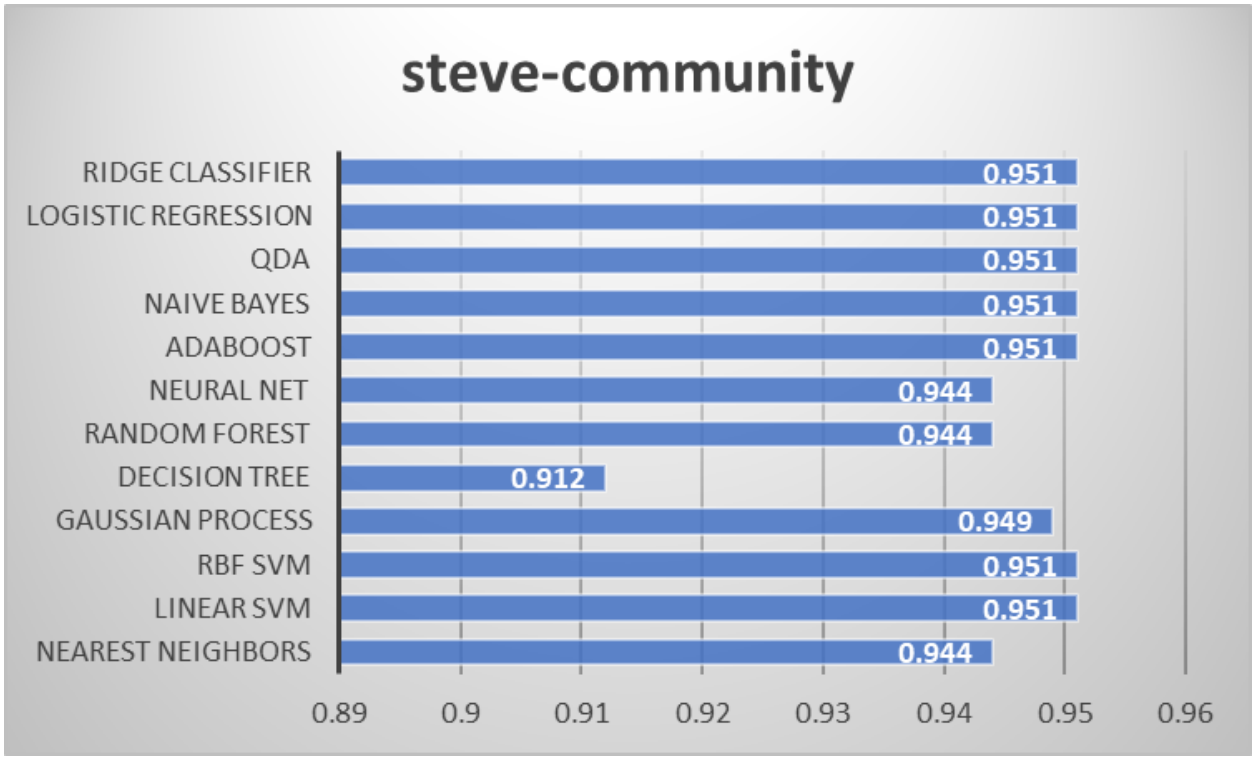

GrammarViz2
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
0.875
0.875
0.911
0.911
0.821
0.884
0.821
0.393
0.884
0.884
0.884
0.813
0
0.2
0.4
0.6
0.8
1

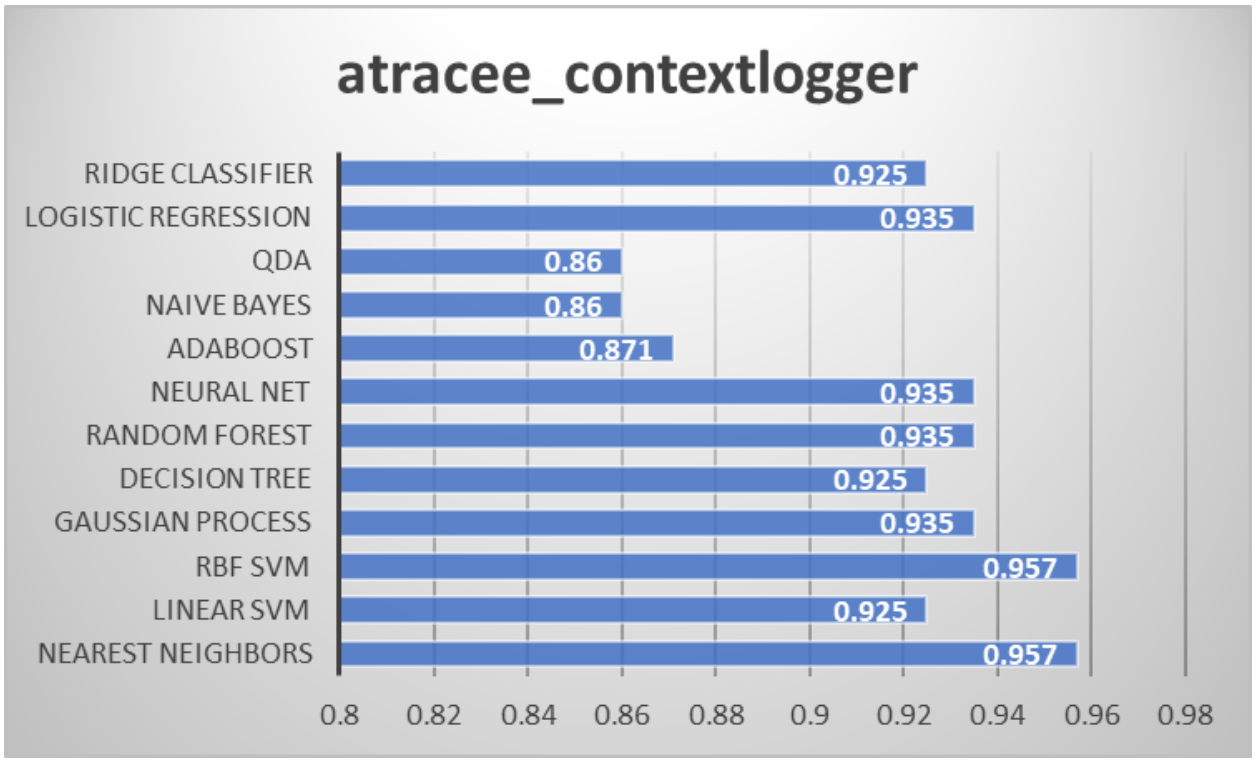

jMotif_GI
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
0.865
0.969
0.875
0.875
0.865
0.969
0.875
0.792
0.969
0.875
0.938
0.875
0
0.2
0.4
0.6
0.8
1
1.2

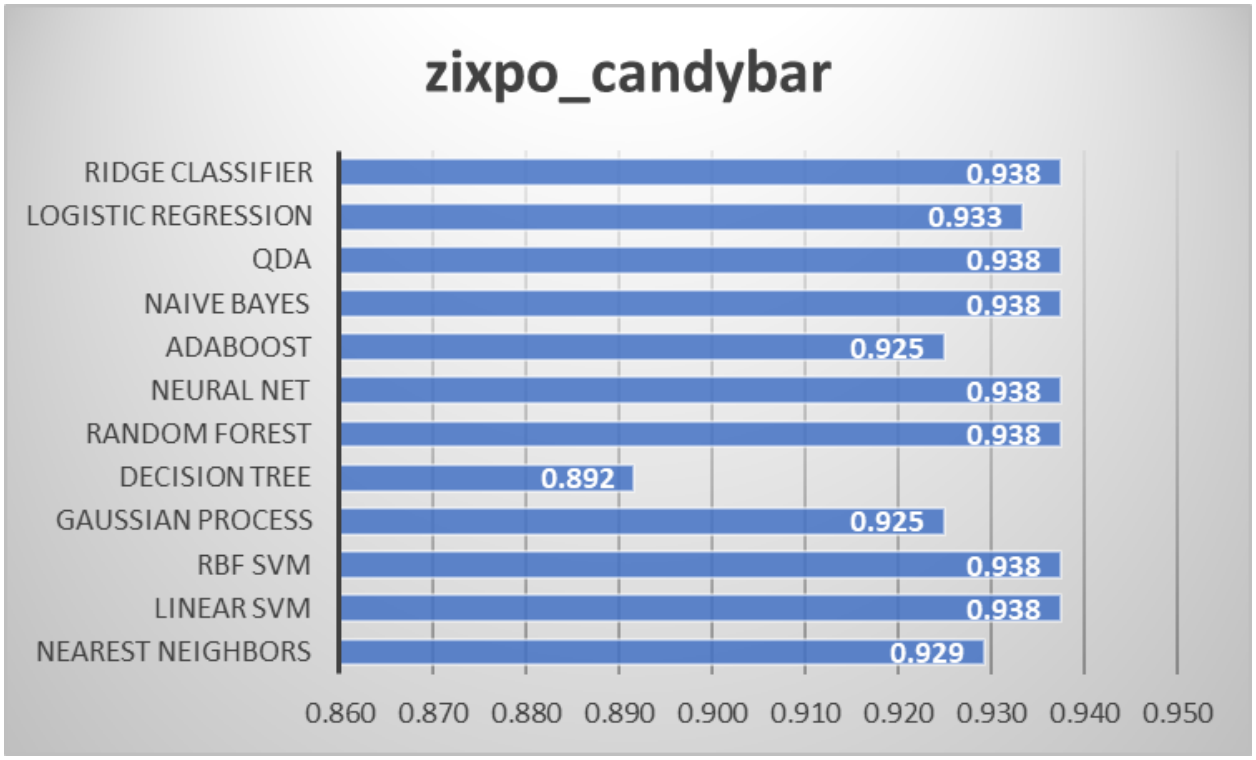

jMotif_SAX
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
0.952
0.952
0.869
0.869
0.869
0.952
0.958
0.958
0.958
0.952
0.952
0.958
0.82
0.84
0.86
0.88
0.9
0.92
0.94
0.96
0.98


ksclarke_solr-iso639-filter
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
0.802
0.802
0.594
0.594
0.557
0.840
0.821
0.811
0.830
0.821
0.821
0.774
0.000 0.100 0.200 0.300 0.400 0.500 0.600 0.700 0.800 0.900


mtsar
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
0.65
0.67
0.67
0.3
0.61
0.64
0.69
0.66
0.67
0.69
0.64
0.67
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8

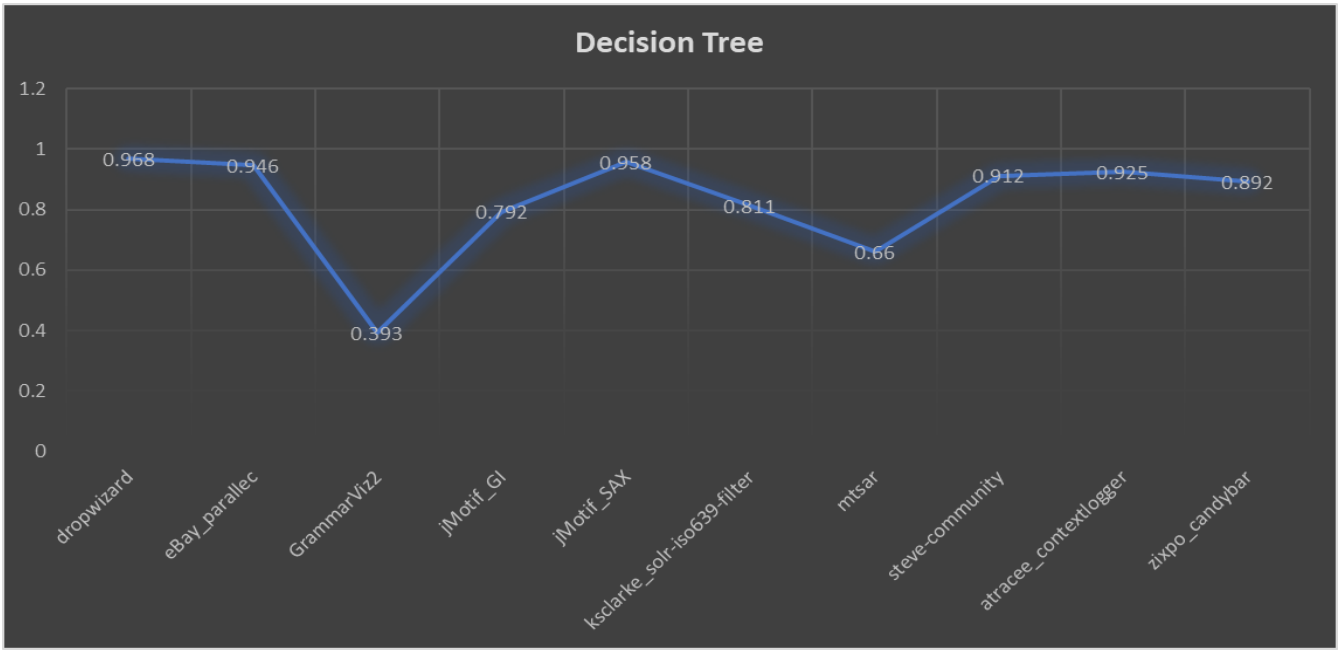

steve-community
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
0.951
0.951
0.951
0.951
0.951
0.944
0.944
0.912
0.949
0.951
0.951
0.944
0.89
0.9
0.91
0.92
0.93
0.94
0.95
0.96


atracee_contextlogger
RIDGE CLASSIFIER
LOGISTIC REGRESSION
QDA
NAIVE BAYES
ADABOOST
NEURAL NET
RANDOM FOREST
DECISION TREE
GAUSSIAN PROCESS
RBF SVM
LINEAR SVM
NEAREST NEIGHBORS
0.925
0.935
0.86
0.86
0.871
0.935
0.935
0.925
0.935
0.957
0.925
0.957
0.8
0.82
0.84
0.86
0.88
0.9
0.92
0.94
0.96
0.98

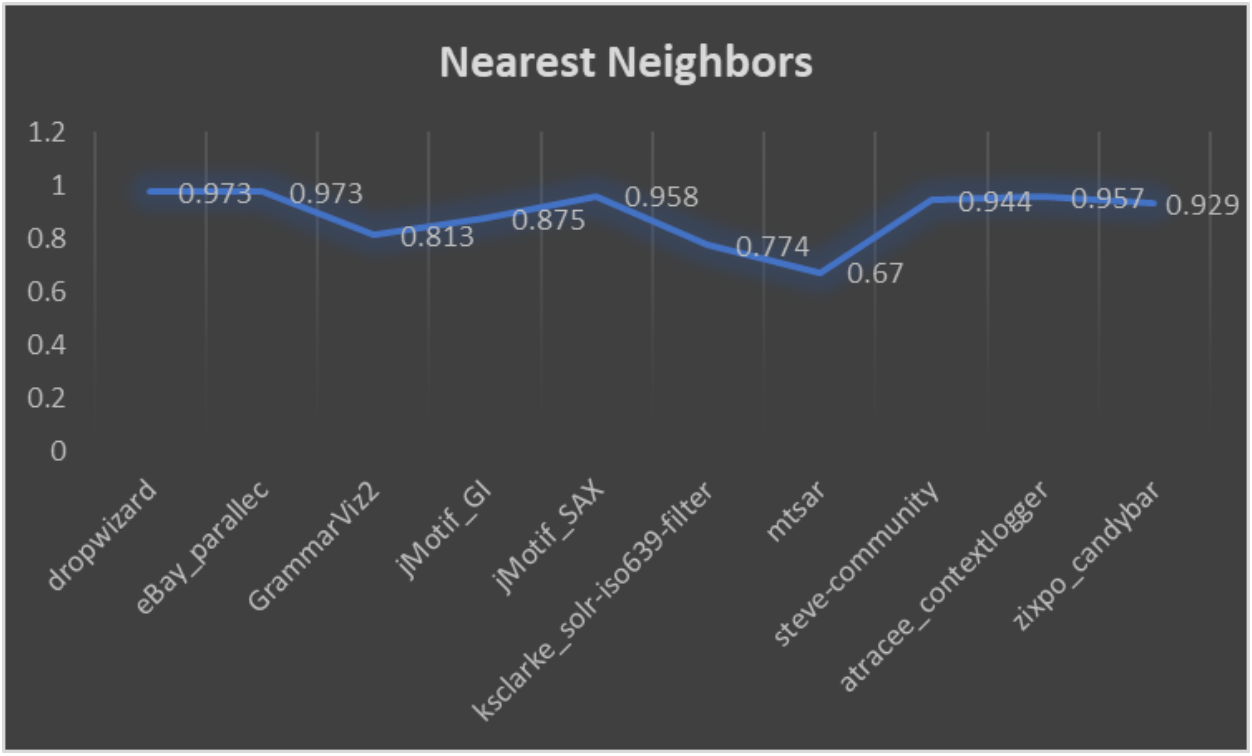


Model wise Performance Trend over Github Repositories

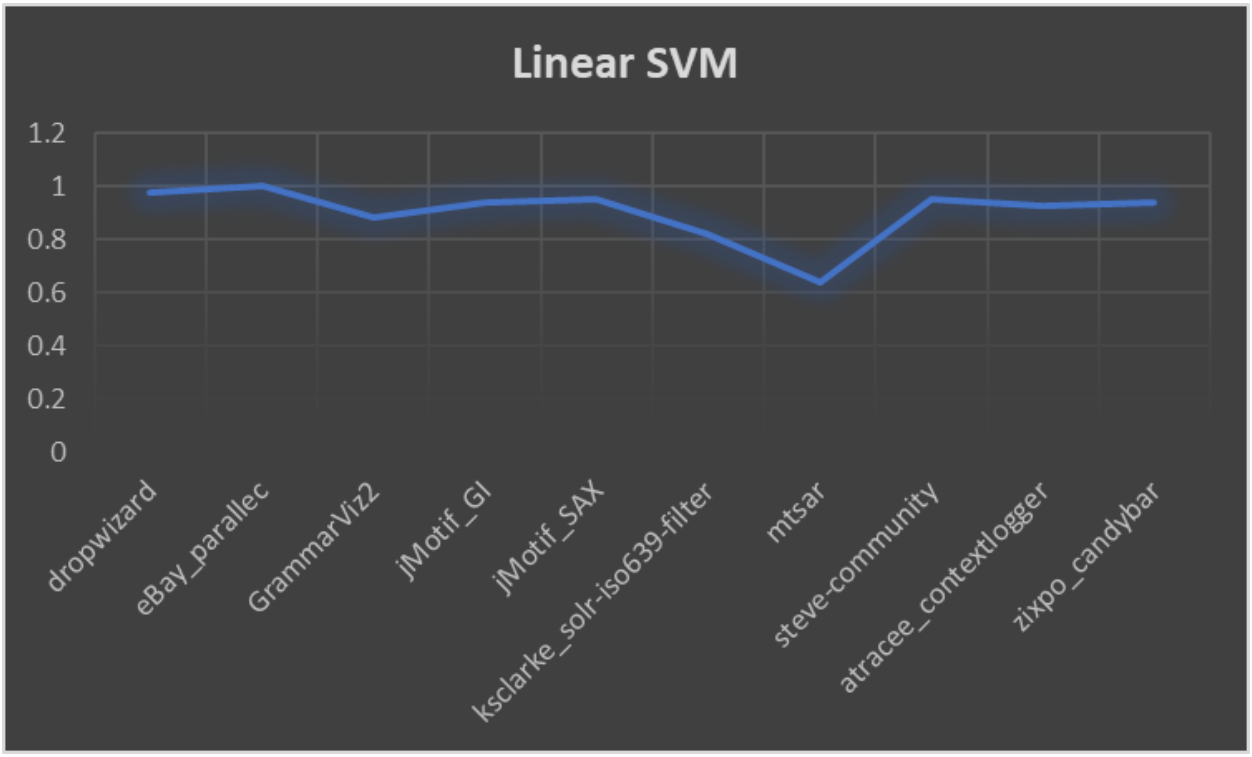
Decision Tree



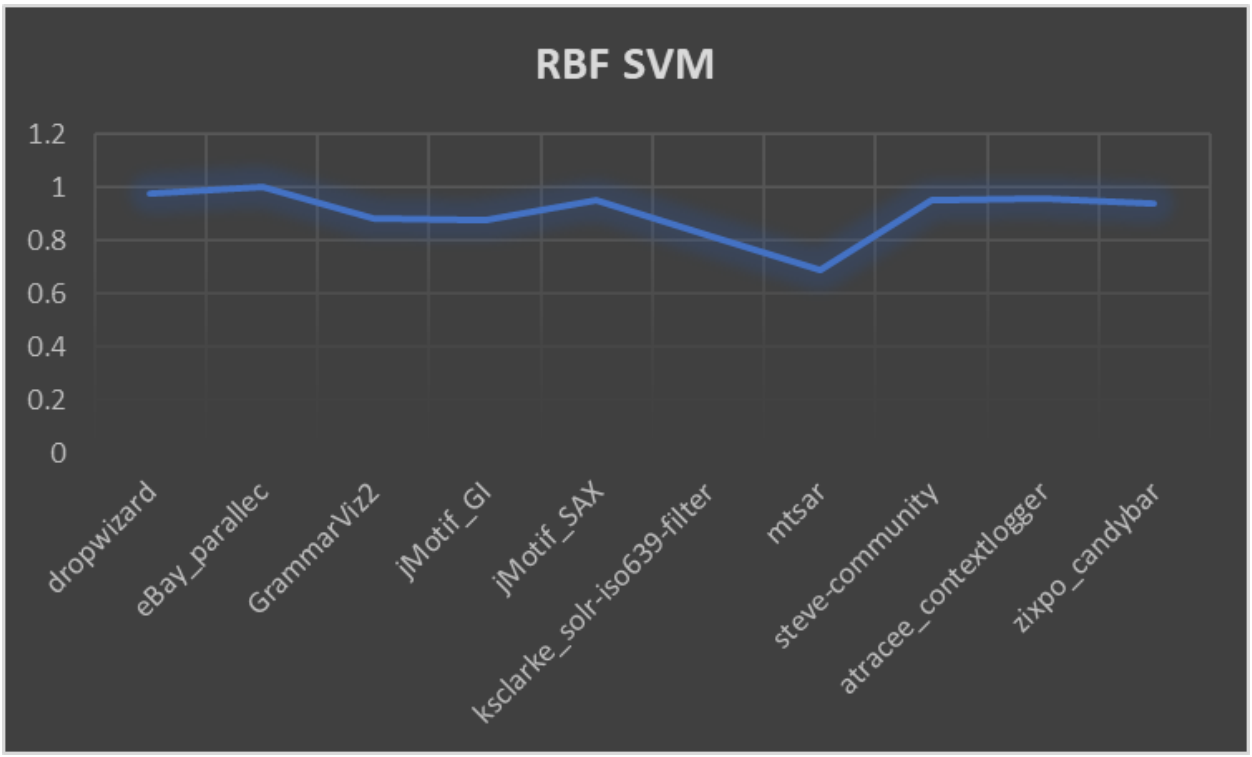
Nearest Neighbors



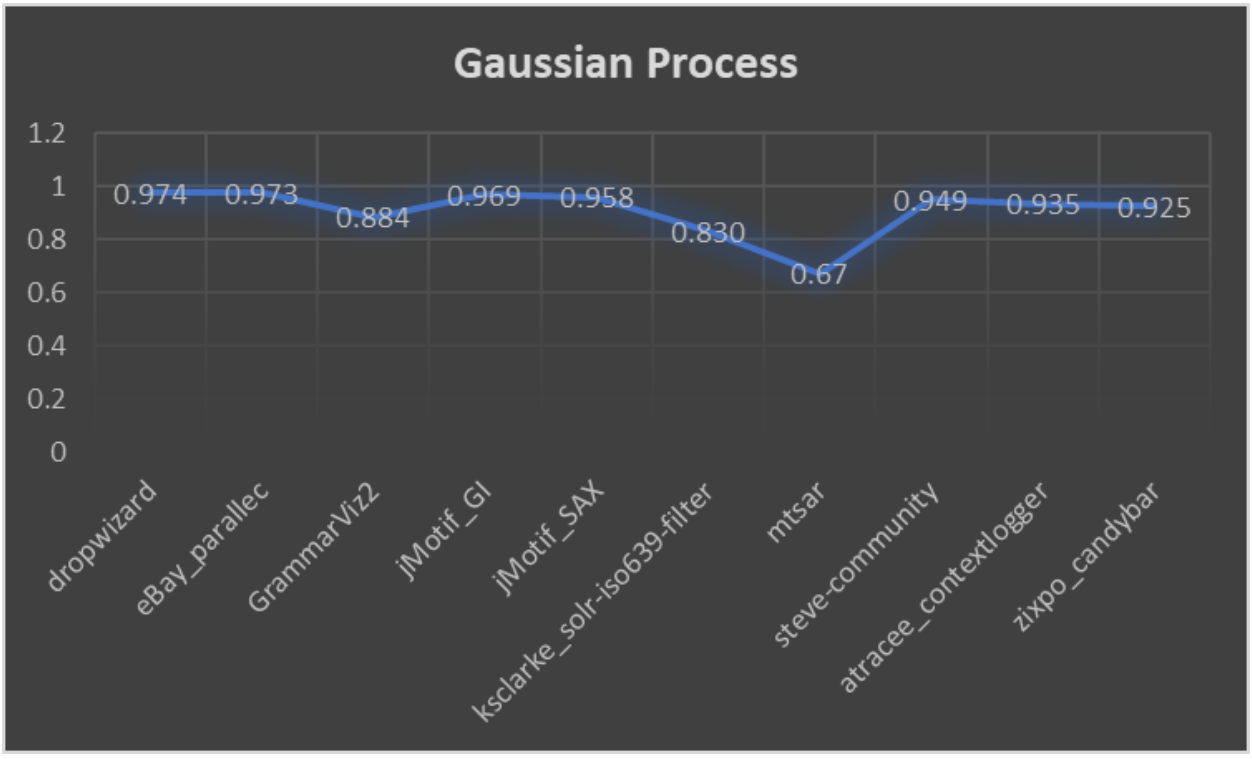
Linear SVM



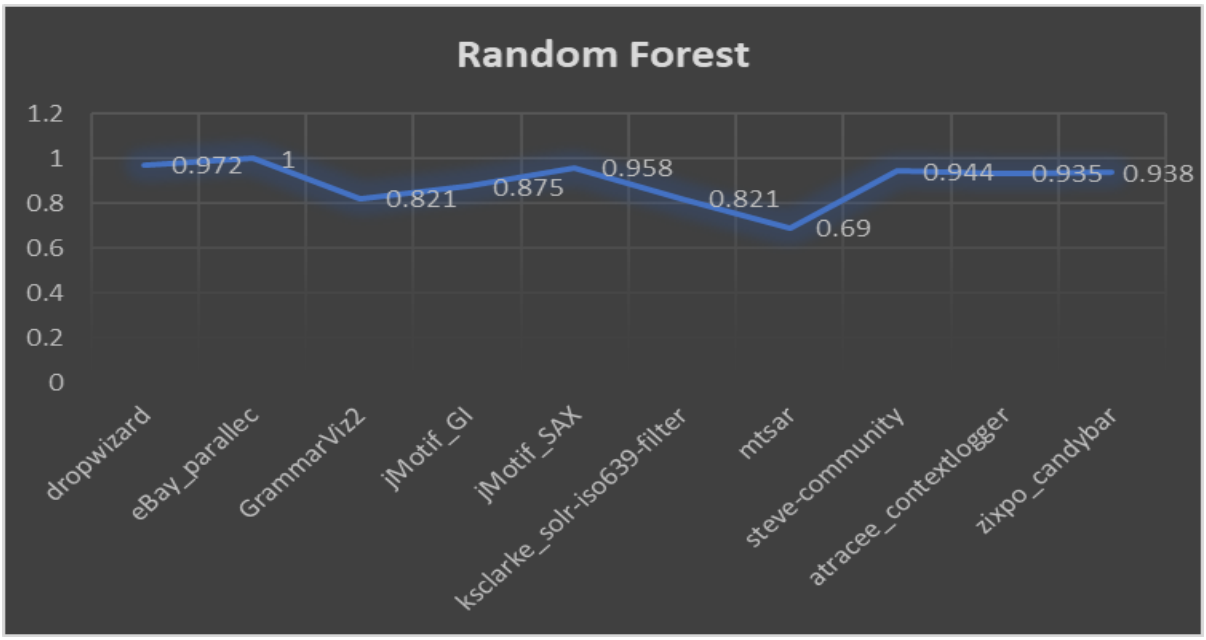
RBF SVM



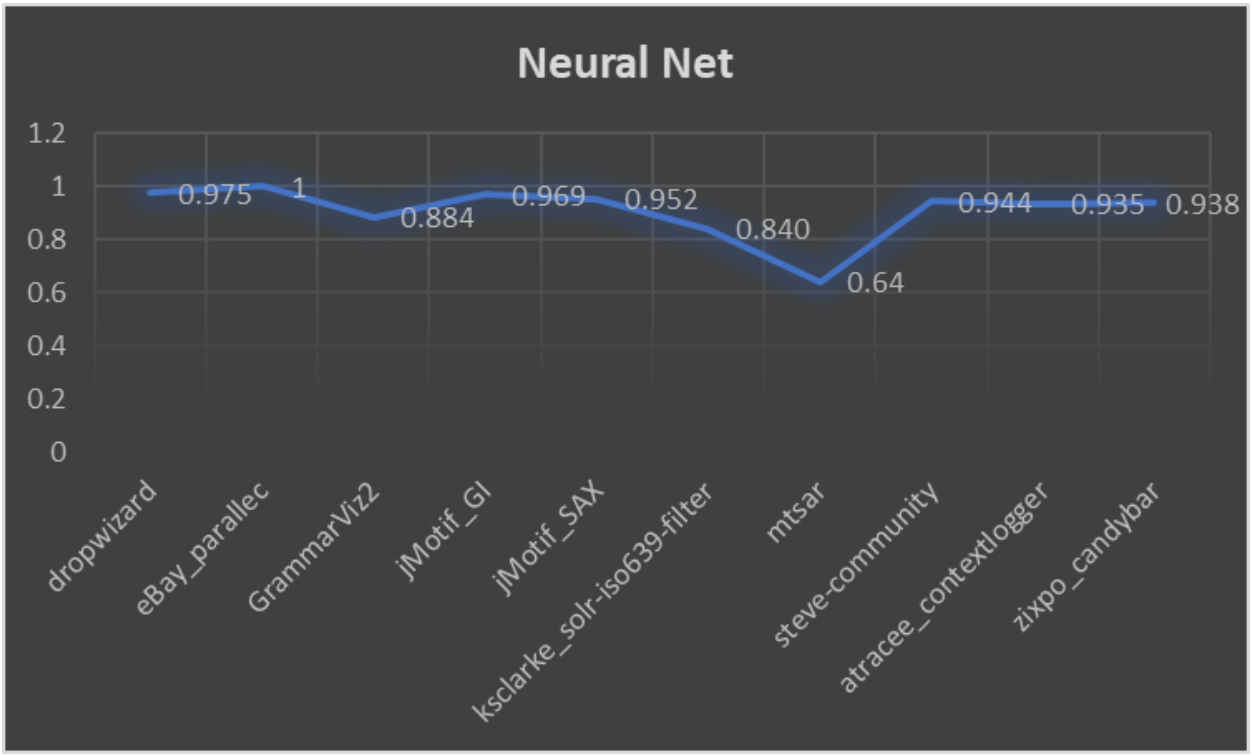
Gaussian Process



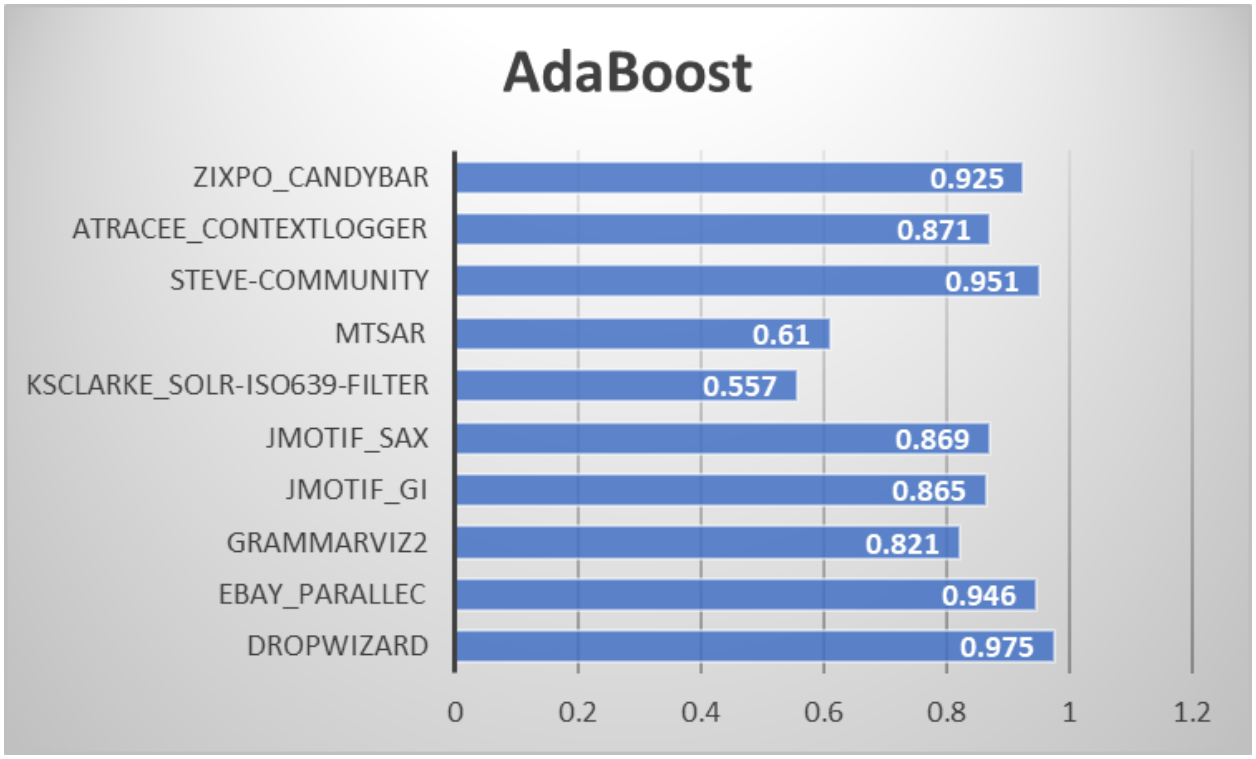
Random Forest



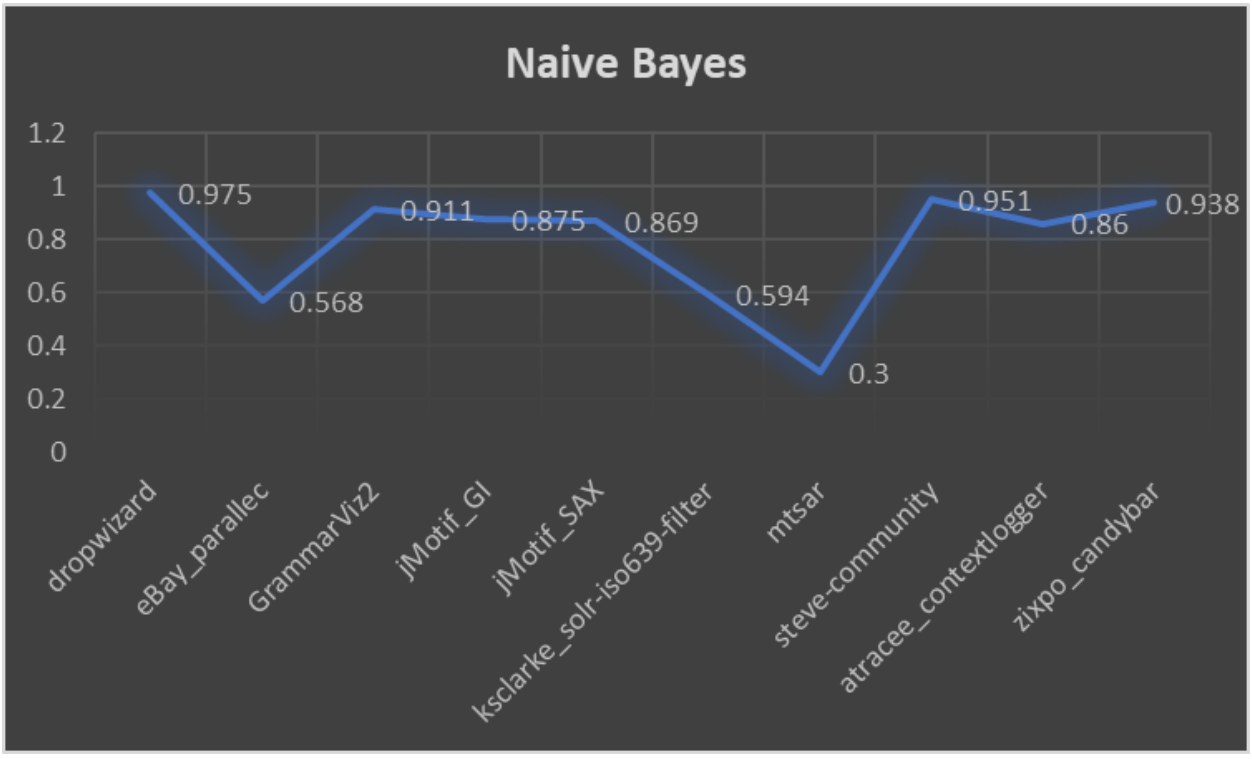
Neural Net


Neural Net
1.2
1
0.8
0.6
0.4
0.2
0
0.975
1
0.884
0.969
0.952
0.840
0.64
0.944
0.935
0.938
dropwizard
eBay_parallec
GrammarViz2
jMotif_GI
jMotif_SAX
ksclarke_solr-iso639-filter
mtsar
steve-community
atracee_contextlogger
zixpo_candybar

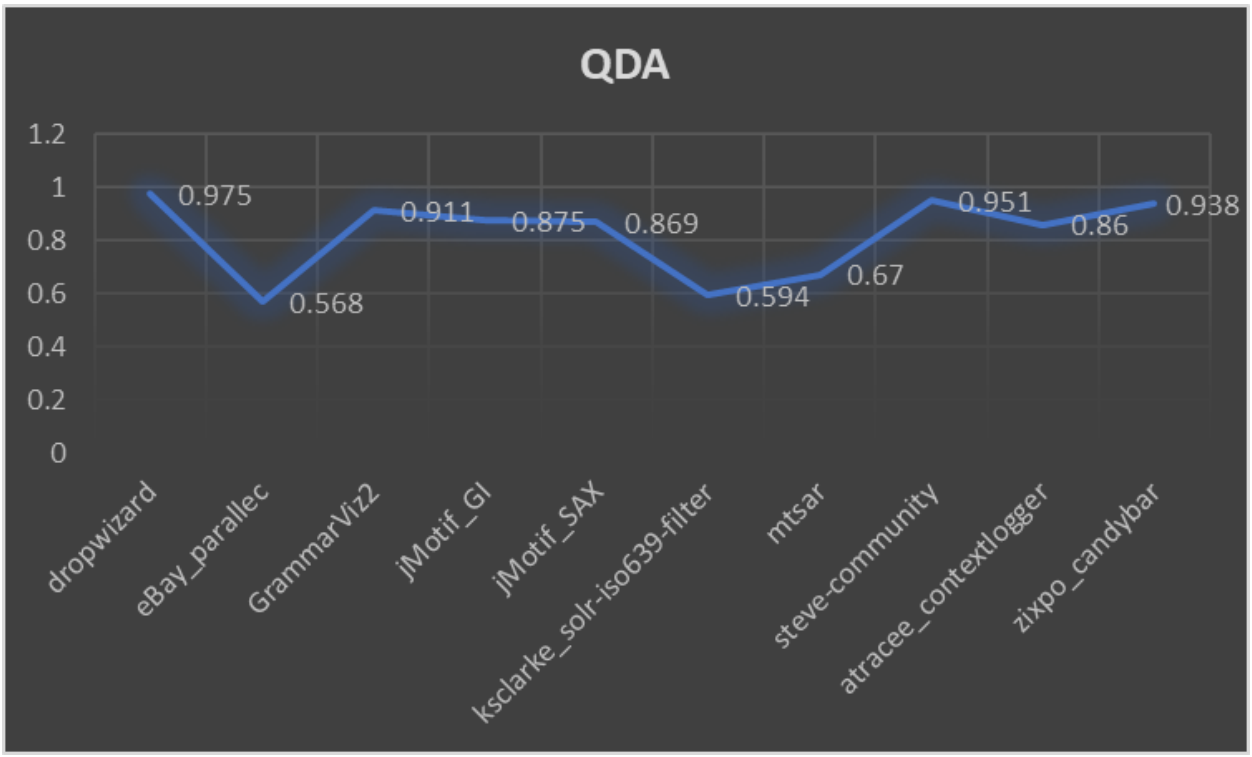
AdaBoost



Naive Bayes

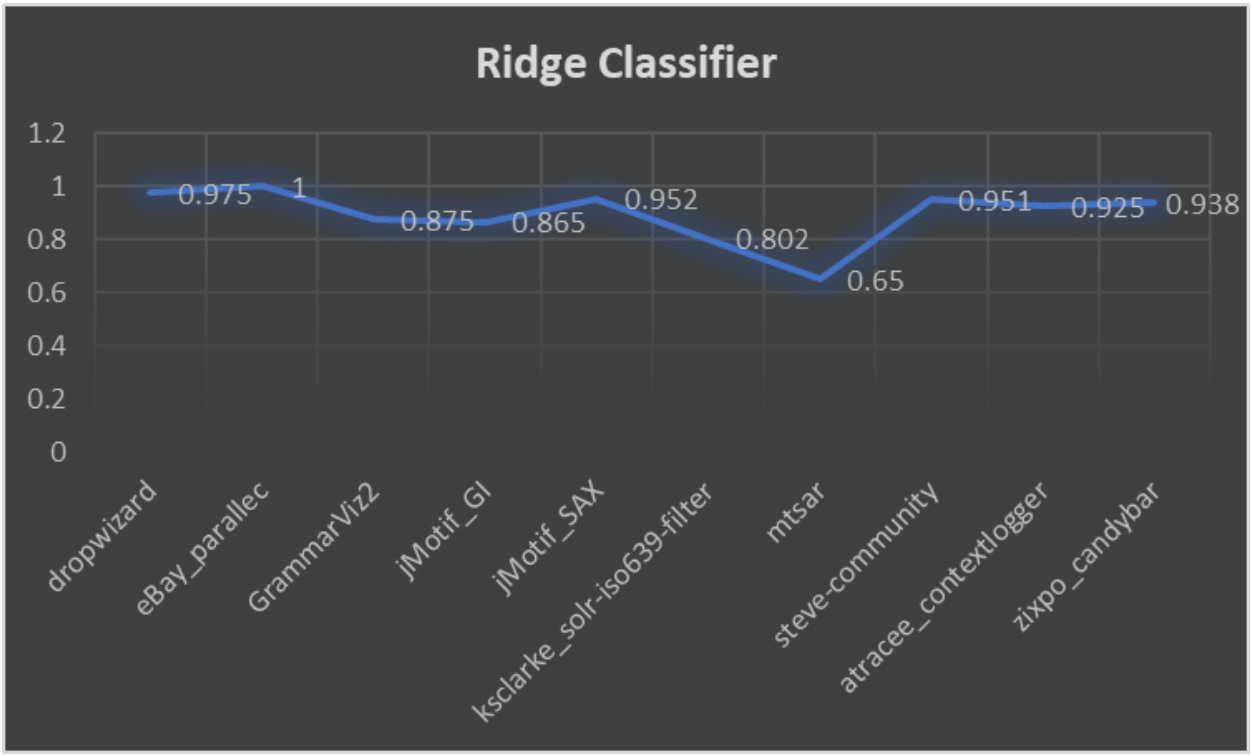


QDA



Logistic Regression

Ridge Classifier



6. Conclusion

Summary of Performance of Different Models on All the Test Repos

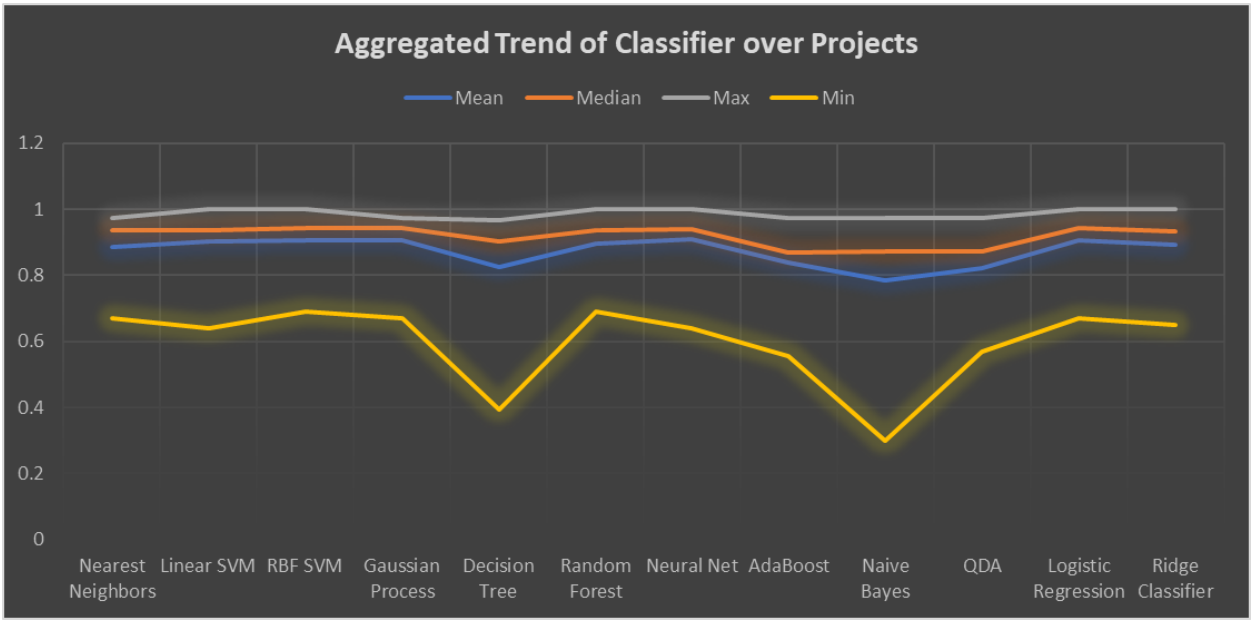
Table 6.1

classifier	dropwizard	eBay_parallec	GrammarViz2	jMotif_GI	jMotif_SAX	ksclarke_solr-iso639-filter	mtsar	steve-community	atracee_contextlogger	zixpo_candybar
Nearest Neighbors	0.973	0.973	0.813	0.875	0.958	0.774	0.67	0.944	0.957	0.929
Linear SVM	0.975	1	0.884	0.938	0.952	0.821	0.64	0.951	0.925	0.938
RBF SVM	0.975	1	0.884	0.875	0.952	0.821	0.69	0.951	0.957	0.938
Gaussian Process	0.974	0.973	0.884	0.969	0.958	0.830	0.67	0.949	0.935	0.925
Decision Tree	0.968	0.946	0.393	0.792	0.958	0.811	0.66	0.912	0.925	0.892
Random Forest	0.972	1	0.821	0.875	0.958	0.821	0.69	0.944	0.935	0.938
Neural Net	0.975	1	0.884	0.969	0.952	0.840	0.64	0.944	0.935	0.938
AdaBoost	0.975	0.946	0.821	0.865	0.869	0.557	0.61	0.951	0.871	0.925
Naive Bayes	0.975	0.568	0.911	0.875	0.869	0.594	0.3	0.951	0.86	0.938
QDA	0.975	0.568	0.911	0.875	0.869	0.594	0.67	0.951	0.86	0.938
Logistic Regression	0.975	1	0.875	0.969	0.952	0.802	0.67	0.951	0.935	0.933
Ridge Classifier	0.975	1	0.875	0.865	0.952	0.802	0.65	0.951	0.925	0.938

Table 6.2

classifier	Mean	Median	Max	Min	Standard Deviation
Nearest Neighbors	0.88657516	0.936583333	0.973	0.67	0.098
Linear SVM	0.90232547	0.93775	1	0.64	0.099
RBF SVM	0.90422547	0.94425	1	0.69	0.088
Gaussian Process	0.90671887	0.942	0.974	0.67	0.090
Decision Tree	0.82569874	0.901833333	0.968	0.393	0.170
Random Forest	0.89532547	0.93625	1	0.69	0.090
Neural Net	0.90761226	0.94075	1	0.64	0.099
AdaBoost	0.83896038	0.87	0.975	0.556604	0.136
Naive Bayes	0.78408396	0.872	0.975	0.3	0.210
QDA	0.82108396	0.872	0.975	0.568	0.144
Logistic Regression	0.90622201	0.943	1	0.67	0.095
Ridge Classifier	0.89323868	0.93125	1	0.65	0.098

Aggregated trend of Performance of Different Models on All the Test Repos



In this project we tried to implement the ML model that can learn the historical commits in any git repository and based on 10-11 features we selected it can learn to skip the commits which are not so important and for such commits we must not run ci-cd pipeline which generally takes much longer. There are now a days feature in CI-CD tools that if you indicate a commit not be marked for CI-CD it will not run but then there are multiple operational problems associated with it, what if the developer is new to team, what if the developer is new to software development process itself in such scenarios it will be difficult for them to actually identify and mark such commits. Also As we can see that our Models have performed equally good as compared to paper and for some of the repos it has out performed but for some of them it has performed on a bit lower side. But if we look at the Table 6.2 the minimum mean value is 0.78 which is still better and not so poor which is for Naive Bayes based ML model which is a probabilistic model so we can say that probabilities models can not perform that great in the problem domain. If we look at the mean which is one of the highest and standard deviation of which is one of the lowest so Neural Net which is the best performing model which was not used by the authors of the project. However since we can not determine which particular pictures are most important so easily and developers can not find which feature model is considered to be most important and give that particular item value so practically in a project using a Neural network won't be a great idea. But just after Neural network and the SVMs which have similar problems as neural networks the next best is Random forest and that can help us find the best feature which is one of the most important factor here to determine the most efficient machine learning model would be random forest which is also recommend in the paper.