Using Hoeffding Bounds and Project Elimination for faster Bellwether prediction

Advisor:
Dr. Timothy Menzies

Akshay Nalwaya
(200159155)

Tasks involved in this project and their owners

- This project work was started as a group project in Fall 2018 for CSC 591: Foundations of Software Science course
- The group members were: Akshay Nalwaya, Sanjana Kacholia, Shantanu Sharma
- Establishing the baseline model, implementing Hoeffding bounds and performing iterative sampling were done collectively by the group members
- I have added project elimination and feature selection strategies to this work

bell·weth·er

/ˈbelˌweTHər/

noun

the leading sheep of a flock, with a bell on its neck.

- an indicator or predictor of something.
  "college campuses are often the bellwether of change"

Motivation

- *Why*: Identifying the Bellwether project among a group of projects would make the task of defect prediction easier
- *What*: Making the identification of this Bellwether project faster than the current $O(N^2)$ approach
- *How*: Using Hoeffding bounds and project elimination to reduce the dataset required for Bellwether identification

Why Bellwethers?

- Model built using Bellwether project can serve as a representative model among the projects in the same domain
- Bellwether project can serve as a baseline model for constructing different transfer learners in various domains of software engineering
- Instead of exploring all the available data, we find one dataset that offers stable results for longer period of time

Existing Approach

Train

- Use a Random Forest Classifier
- Train on data from one of the projects at a time, for all projects

Test

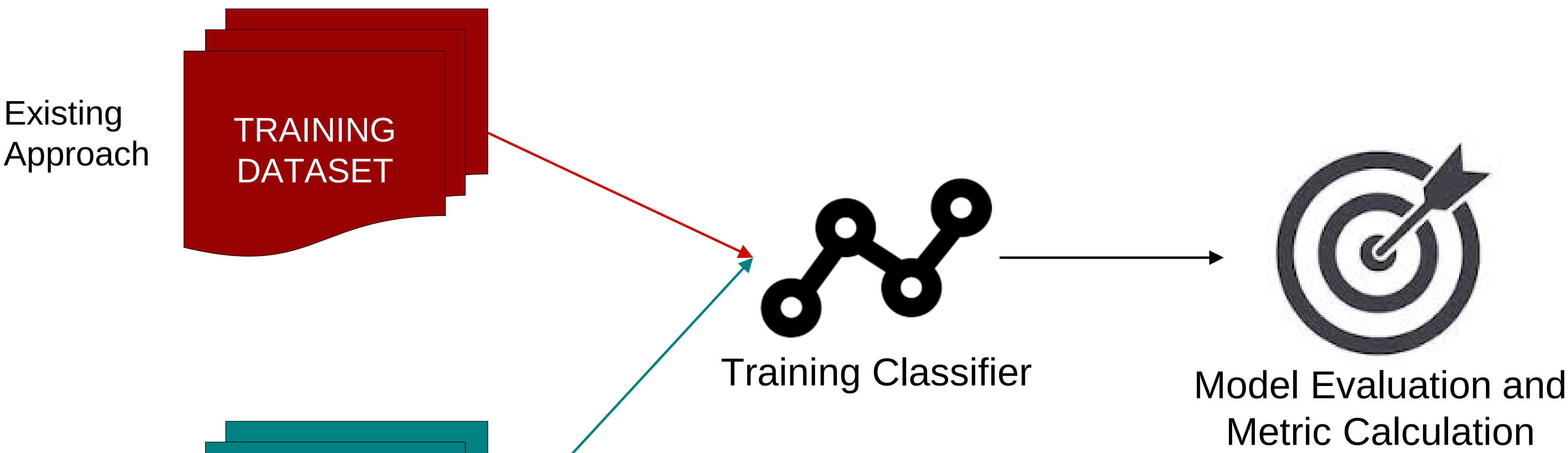
- Test on all the other projects
- Compute *g-score* for each project

$$G = \frac{2 * pd * (1 - pf)}{(1 + pd - pf)}$$

Bellwether Prediction

- Calculate median *g-score*
- Project with best median *g-score* is declared as *"Bellwether"*

New approach reduces the amount of data used for training classifier



Advantages:

- Reduction in the model building time, hence reduced space and time requirement
- Faster identification of Bellwether

Research Questions

RQ1 : Can we predict which dataset will be the bellwether?

RQ2 : Can we reduce the time to find bellwether by reducing the size of data?

RQ3 : Does Hoeffding sampling give better performance than project elimination?

RQ4 : Does feature selection improve the time for bellwether identification?

Sampling using Hoeffding Bounds

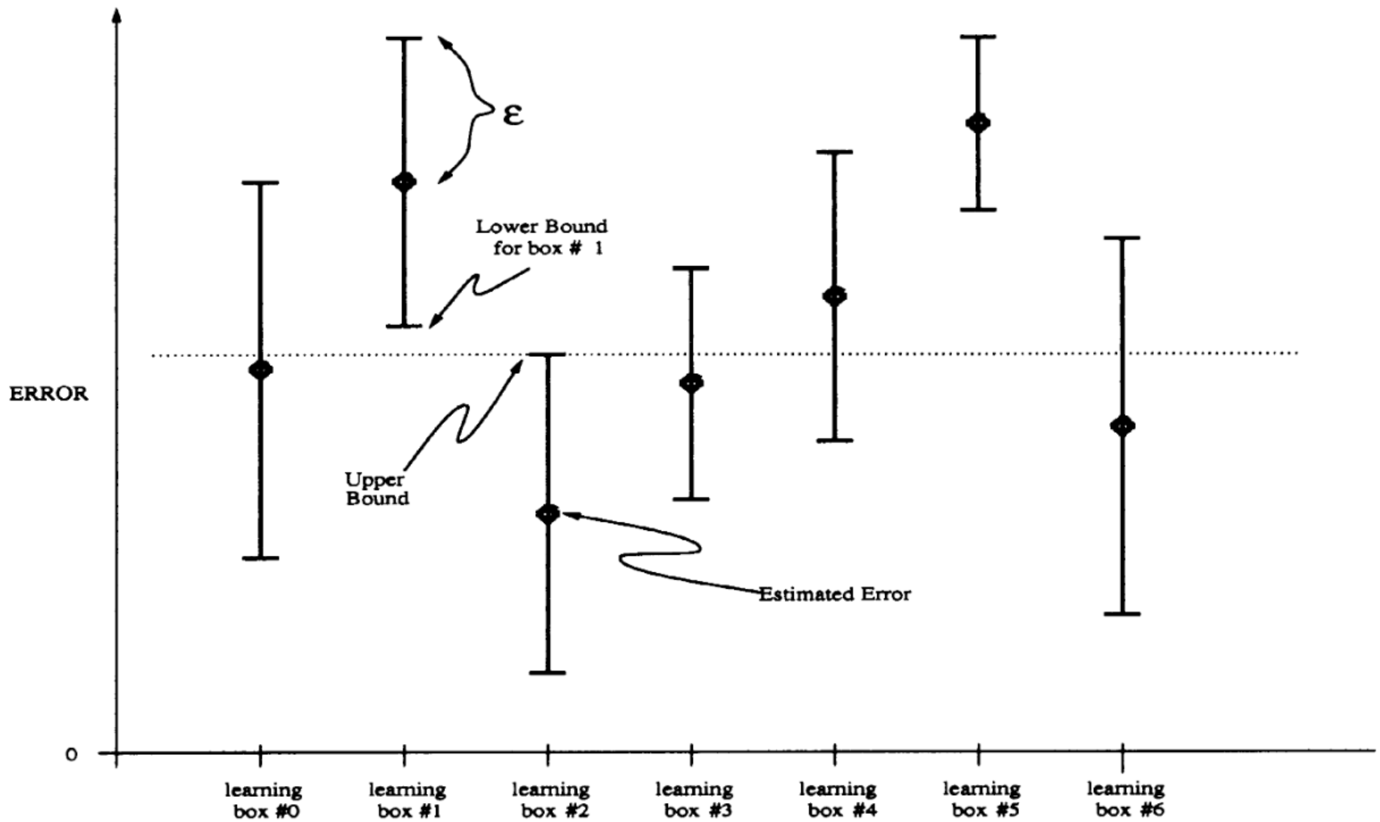
- Iteratively keep on adding data points from the data till a sufficient number of points have been picked
- Finding how close is estimated error from true error

$$\Pr(|E_{true} - E_{est}| > \epsilon) < 2e^{-2n\epsilon^2/B^2}$$

- We estimate the number of samples required using

$$n > \frac{B^2 \log(2/\delta)}{2\epsilon^2}$$

Sampling using Hoeffding Bounds



The upper bound of learning box #2 eliminates the learning boxes #1 and #5

Project Elimination to reduce the candidate projects

- Core idea behind this approach is to eliminate projects having significantly poor performance from the pool of candidate projects
- This will reduce the number of projects required to be analyzed for making bellwether identification
- Conditions for elimination:
  - Project is testing on atleast 1/3rd of the projects
  - G-score value is less than the threshold value

Project Elimination to reduce the candidate projects

Algorithm

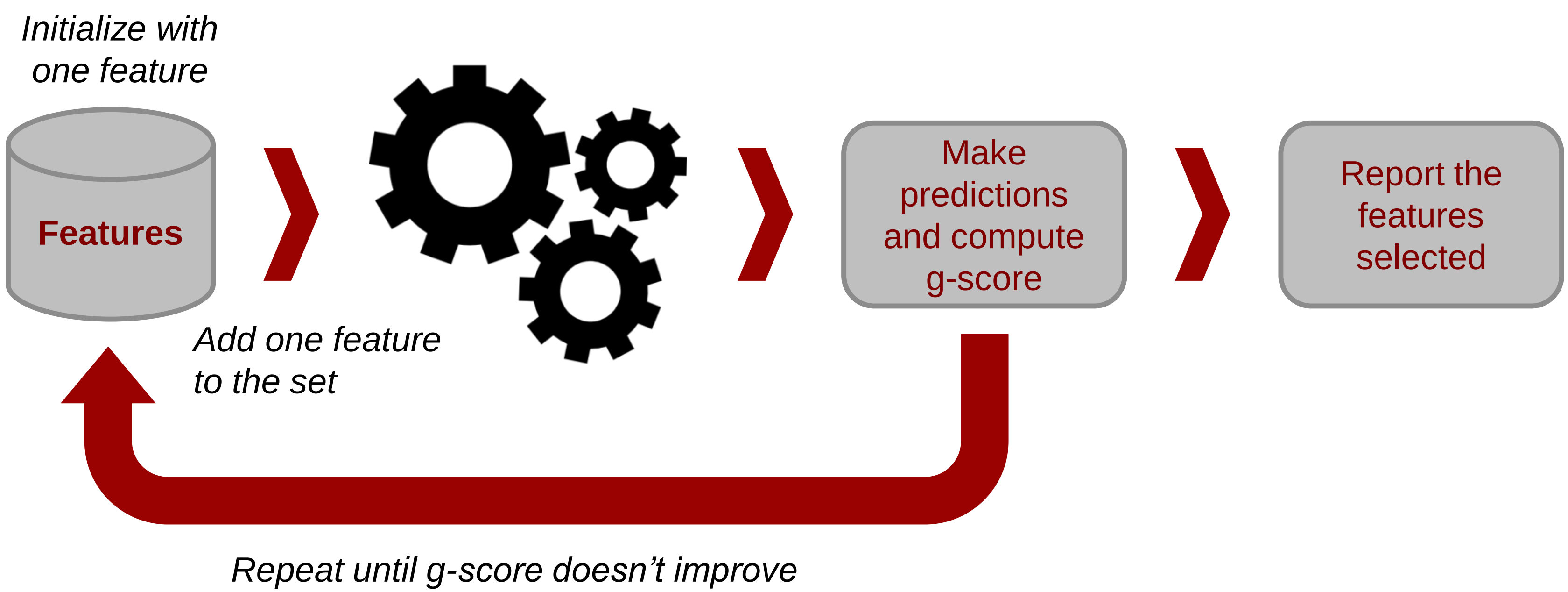
```
for each project do
      load X_train, y_train
      train random forest classifier
      set threshold g-score
      for all other projects
            load X_test, y_test
            make predictions
            compute g-score
            if g-score < threshold and #projects tested >= 3:
                  g-score = 0
            append results, g-score
return results
```

Exploring Feature Selection algorithms to filter unimportant attributes

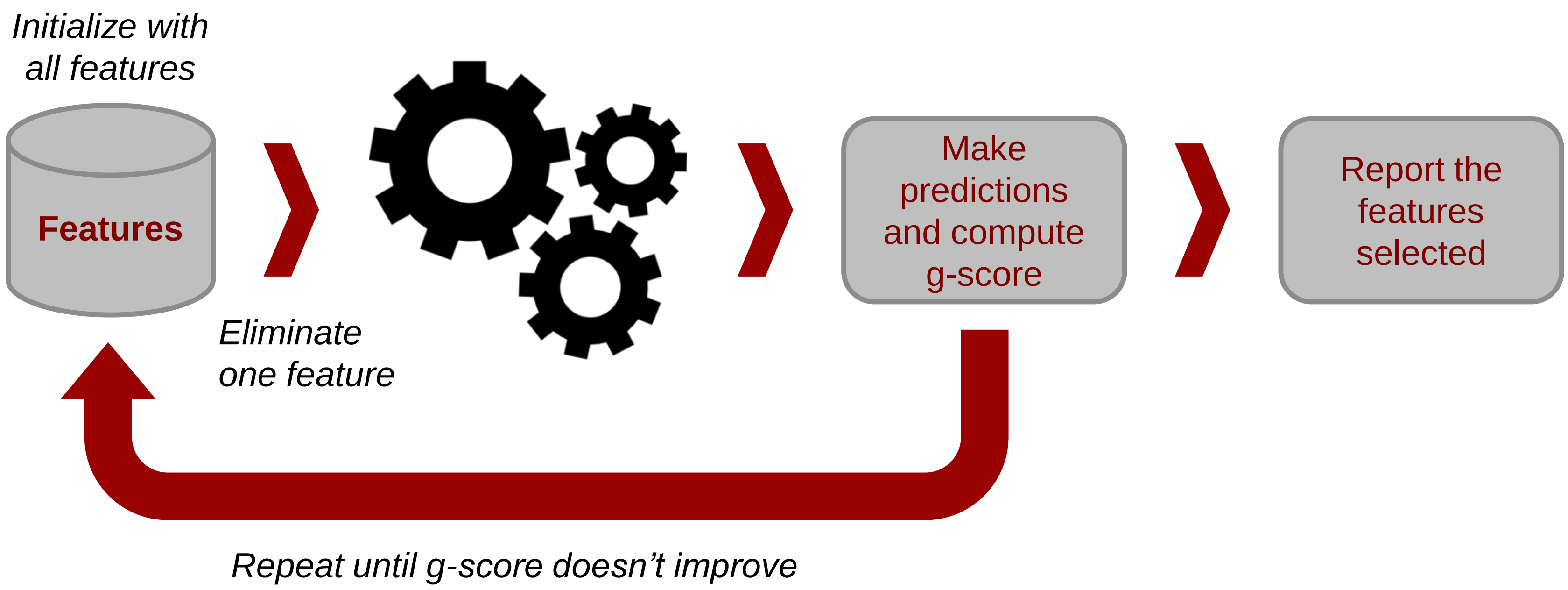
In this work, we have explored the following feature selection algorithms:

- Forward feature selection
- Backward feature elimination
- Information Gain as a feature selector
- Correlation as a feature selector

Forward Feature Selection



Backward Feature Elimination



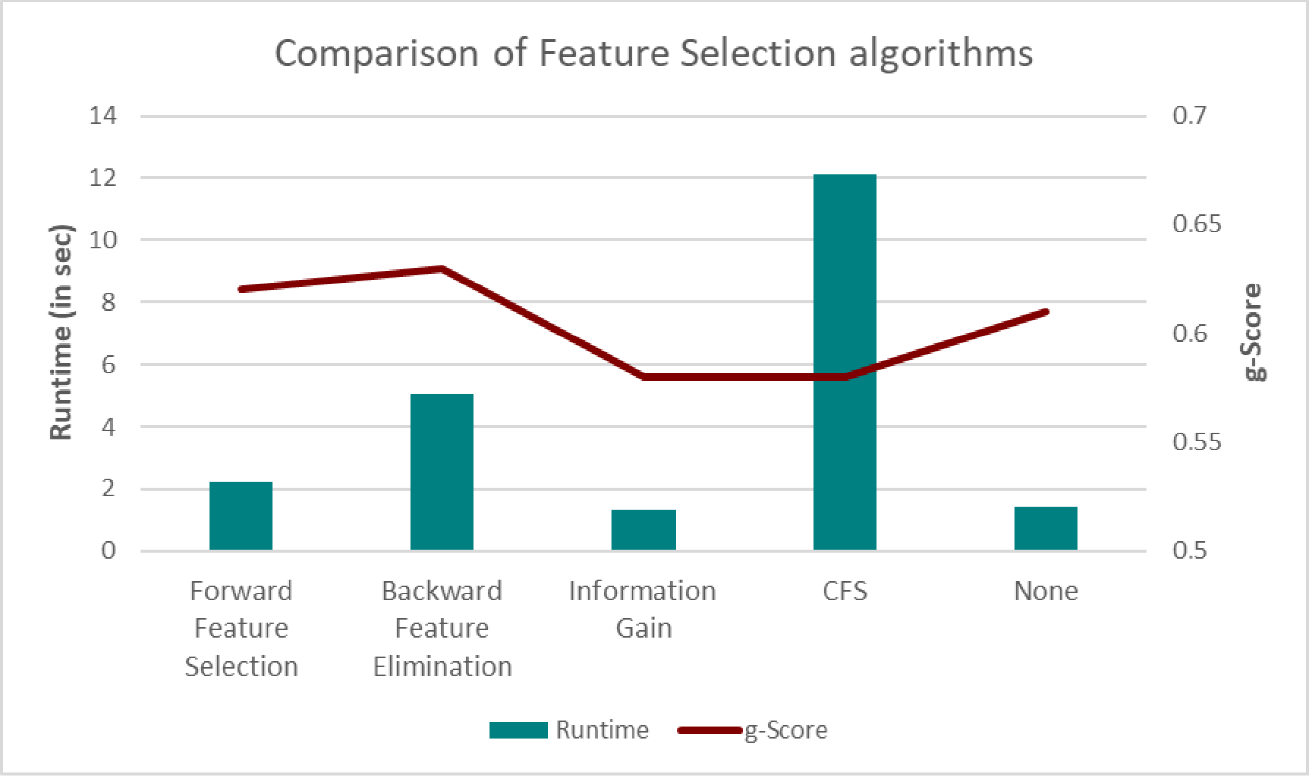
Information Gain as a feature selector

- Entropy-based feature evaluation method
- *Information Gain:* Amount of information provided by a feature for the items to be predicted
- Information gain for each attribute is calculated and attributes with higher values of information gain are chosen
- Attributes with lower information gain are eliminated since they do not provide significant information about the class label

Correlation-based Feature Selection (CFS)

- Evaluates subsets of attributes rather than individual attributes
- Considers the usefulness of individual attributes for predicting class label and also the inter-correlation between attributes
- *Ideal subset:* High correlation with class while low inter-correlation with each other
- Computes correlation between attributes and applies heuristic search strategy for finding ideal subset

Comparison of these feature selection algorithms



NOTE: These results are for the Bellwether project (poi)

Key takeaways from the feature selection approaches

- None of the approaches provide a significant improvement of g-score than conventional Random Forest Classifier
- Forward selection and Backward elimination methods operate in a sequential manner and hence do not cover all subsets
- Information gain takes each attribute but does not account of relationship between attributes
- CFS answers the shortcomings of other approaches but takes a lot of time to run without any proportional improvement

Comparing results from Hoeffding bounds and Project Elimination

Project	Baseline Approach	Hoeffding Bounds	Modified Hoeffding Bounds
ant	0.18	0.18	0.19
camel	0.24	0.25	0.24
ivy	0.09	0.12	0.12
jedit	0.04	0.03	0.04
log4j	0.34	0.34	0.32
lucene	0.52	0.52	0.51
poi	**0.61**	**0.62**	**0.61**
velocity	0.49	0.49	0.49
xalan	0.56	0.58	0.57
xerces	0.42	0.43	0.43

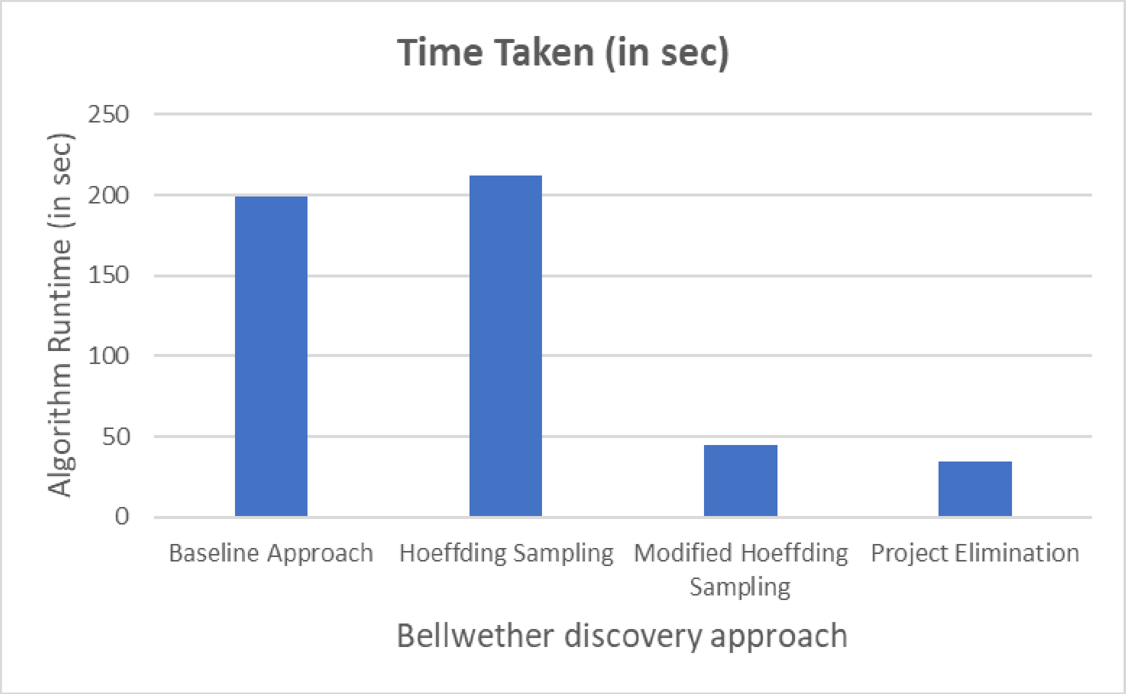
- ***'poi'*** is the bellwether dataset for the baseline method as well as after the implementation of Hoeffding bounds.
- Training data of around ~8.5% for each dataset gives similar results, **reducing the time and data required for training effectively**.

Project elimination performance

Project	Baseline Approach	Project Elimination
ant	0.18	0.0
camel	0.24	0.0
ivy	0.09	0.0
jedit	0.04	0.0
log4j	0.34	0.0
lucene	0.52	0.54
poi	**0.61**	**0.61**
velocity	0.49	0.49
xalan	0.56	0.57
xerces	0.42	0.41

Key takeaways:
- ***'poi'*** remains the bellwether project for this approach also
- Projects which are pruned are assigned value 0
- G-score values are very close to those obtained by the baseline approach

Average runtime for all the approaches for Bellwether identification



Experiment	Runtime (in sec)
Baseline Approach	199.32
Hoeffding Sampling	211.57
Modified Hoeffding Sampling	44.16
Project Elimination	34.63

Future Work / Open issues

- Exploring alternative sampling techniques
- Extending this work to different target domains like code smells, issue lifetime estimation and effort estimation
- Racing between project elimination and sampling

APPENDIX

Metrics for measuring classifier performance

		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

$$accuracy = \frac{true\ positive + true\ negative}{total\ number\ of\ instances}$$

$$recall\ (or\ pd) = \frac{true\ positive}{true\ positive + false\ negative}$$

$$pf = \frac{false\ positive}{false\ positive + true\ negative}$$

$$G = \frac{2 * pd * (1 - pf)}{(1 + pd - pf)}$$

Data Dictionary

Metric Notation	Metric Name	Metric Description
$amc	Average Method Complexity	This metric measures the average method size for each class. Size of a method is equal to the number of Java binary codes in the method.
$avg_cc	Average of Cyclomatic Complexity (CC)	CC is equal to number of different paths in a method (function) plus one. The McCabe cyclomatic complexity is defined as: CC=E-N+P; where E is the number of edges of the graph, N is the number of nodes of the graph, and P is the number of connected components. CC is the only method size metric. The constructed models make the class size predictions. Therefore, the metric had to be converted to a class size metric.
$ca	Afferent Coupling	The CA metric represents the number of classes that depend upon the measured class.
$cam	Cohesion Among Class Methods	This metric computes the relatedness among methods of a class based upon the parameter list of the methods. The metric is computed using the summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.
$cbm	Coupling between Methods	The metric measures the total number of new/redefined methods to which all the inherited methods are coupled. There is a coupling when at least one of the conditions given in the IC metric is held.
$cbo	Coupling between Object Classes	The CBO metric represents the number of classes coupled to a given class (efferent couplings and afferent couplings).
$ce	Efferent Couplings	The CE metric represents the number of classes that the measured class is depended upon.
$dam	Data Access Metric	This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
$dit	Depth of Inheritance Tree	The DIT metric provides for each class a measure of the inheritance levels from the object hierarchy top.
$ic	Inheritance Coupling	This metric provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods functionally dependent onthe new or redefined methods in the class.

Data Dictionary

Metric Notation	Metric Name	Metric Description
$lcom	Lack of cohesion in methods	The LCOM metric counts the sets of methods in a class that are not related through the sharing of some of the class fields.
$lcom3	Lack of cohesion in methods	A low value of LCOM2 or LCOM3 indicates high cohesion and a well-designed class. It is likely that the system has good class subdivision implying simplicity and high reusability. A cohesive class will tend to provide a high degree of encapsulation. A higher value of LCOM2 or LCOM3 indicates decreased encapsulation and increased complexity, thereby increasing the likelihood of errors.
$loc	Lines of Code	The LOC metric calculates the number of lines of code in the Java binary code of the class under investigation.
$max_cc	Maximum value of CC	The greatest value of CC among methods of the investigated class.
$mfa	Measure of Functional Abstraction	This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.
$moa	Measure of Aggregation	This metric measures the extent of the part-whole relationship, realized by using attributes. The metric is a count of the number of class fields whose types are user defined classes.
$noc	Number of Children	The NOC metric simply measures the number of immediate descendants of the class.
$npm	Number of Public Methods	The NPM metric counts all the methods in a class that are declared as public.
$rfc	Response for a Class	The RFC metric measures the number of different methods that can be executed when an object of that class receives a message.
$wmc	Weighted methods per class	The value of the WMC is equal to the number of methods in the class (assuming unity weights for all methods).

Hoeffding's Bound

$$\epsilon > \sqrt{\frac{B^2 \log(2/\delta)}{2n}}$$

$$n > \frac{B^2 \log(2/\delta)}{2\epsilon^2}$$

where,

n = number of points picked in the current iteration
ε = possible error value
$1 - \delta$ = confidence interval (95% in this case)
B = maximum error that classifier can make
[B = 1, since it's classification problem]

Terminating Conditions:

- *If* $g_{est} \geq g\ from\ baseline\ scores$
- *If* g_{est} *is within the bounds*
- $n = N$