Statistical Learning and Stochastic Processes
Assignment 2 (Part B: Data Analysis)

Instructions

This assignment is to be completed in teams of two students. To offer you an optimal learning experience it is not allowed to use generative AI to perform the actual data analysis. The assignment was designed to be within the scope of the abilities of students who have the required background knowledge for this course and studied the course material. Use of gen AI to learn more about the analysis methods to be applied is of course allowed. In any case, your report should include a separate section stating for what purposes (if any) and to what extent you used gen AI to complete the assignment. You may be invited to answer questions in person about your report afterwards.

This is part B (data analysis) of the second assignment. Part A (statistical inference) has been released on December 19, 2025. Both parts should be handed in together in a single report in pdf format, ultimately on Friday, January 23, 2026.

Introduction

Software quality prediction is an important application of data-driven modeling in computer science. A trained prediction model can be used to automatically flag high-risk code changes during development, helping teams prioritize testing and code review efforts where bugs are most likely to occur. It can also support resource planning by estimating defect rates early, improving release stability and reducing maintenance costs.

In this assignment, you will work with a real dataset containing change metrics and code complexity measures to build predictive models to estimate the number of software bugs in a codebase. By exploring the relationship between how code evolves and how complex it is, your model will help identify patterns that may lead to defects. This task will challenge you to apply data preprocessing, model training, and evaluation techniques to produce meaningful insights and reliable predictions.

We will analyze data available from `https://bug.inf.usi.ch/`. We will use data from the Equinox framework available from the download page, in particular:

- Change metrics (15) plus categorized (with severity and priority) post-release defects.
- CK and other 11 object oriented metrics for the last version of the system plus categorized (with severity and priority) post-release defects.

The two datasets should be combined to obtain a dataset that contains both change metrics and code complexity measures for a total of 324 classes of the Equinox framework. The target variable is `bugs` which is the number of post-release bugs for the class concerned. After removing all columns that should not be used for prediction, the dataset should contain 32 features in total, and 1 target variable representing the number of bugs. For more information on this data, please refer to [1] and references therein.

Regression for Count Data: Poisson Regression

We start by building a model that predicts the number of bugs. Since we are dealing with count data, we apply a model that is suited for this type of data, namely Poisson regression. We first create a training set containing 67% of the data (rows), and a test set containing the remaining 33%. To balance the distribution of bugs in the training and test set, use stratified random sampling from the bins given in Table 1.

Bugs	Frequency
0	195
1-2	105
3-5	19
6+	5

Table 1: Bins to be used for stratified sampling and the frequency for each bin. For example, there are 195 software classes (rows) with 0 bugs, 105 with 1 or 2 bugs, etc.

We will use the training set to select a model, and the test set to estimate the error of the selected model. We compare 2 different model selection methods:

1. Stepwise regression: starting from some initial model (e.g. the model containing only the intercept), perform a local search. Neighboring models are models with one feature added to the current model (forward), one feature removed from the current model (backward), or either of these. To score models, one can use AIC/BIC or out-of-sample predictive performance using cross validation. The local search moves to the best neighbor, unless all neighbors are worse than the current model, in which case we return the current model.

2. Regularization with LASSO (L1) penalty: to avoid overfitting, a penalty for the size (absolute value) of the model coefficients is added to the negative log-likelihood:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ -\ell(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{d} |\beta_j| \right\},$$

   where $-\ell(\boldsymbol{\beta})$ is the negative log-likelihood score, and λ is a tunable parameter that determines the strength of the penalty for the coefficient sizes. The best value for λ is typically determined through cross validation. LASSO (L1) regularization has the property that some coefficients will be shrunken to exactly zero, making it effectively a feature selection method.

Perform model selection on the training set using both these methods. You are allowed to include interaction terms or any other transformation of the original features you see fit. Be aware however that all model selection should use the training data only, and only the model finally selected is evaluated on the test set. This will be the final result you report.

Report both the selected model and its predictive performance on the test set. Discuss the final model in terms of the features included, interpretation of their coefficients and any other information you think is noteworthy. Compare the results obtained with stepwise regression and LASSO regularization.

Binary Classification: Logistic Regression

For some purposes one might be interested in predicting whether or not a piece of code contains bugs rather than predicting the exact number of bugs. For this purpose, we create a binary target variable `bugbin` that takes the value 1 if the code contains one or more bugs, and the value 0 if the code is bug free. Repeat the analysis you performed before, but now use logistic regression instead of Poisson regression, and use predictive performance measures suitable for classification. You can keep the same training set and test set.

Report

Describe the analysis you performed in such a way that the interested reader would be able to reproduce your results. You don't need to write a full paper with introduction, conclusions, etc. It is sufficient to have separate sections for Poisson regression and logistic regression, with subsections for the two model selection methods, and a separate subsection for their comparison. If you perform substantial pre-processing of the data before analysis, you can also include a separate section for data pre-processing.

Software

This is a data analysis assignment, not a coding assignment. You are allowed to use any `R` or `Python` packages you want.

In `R`, stepwise model selection with AIC or BIC scoring can be performed using the function `stepAIC` from the `MASS` package. LASSO regularization can be performed using the function `cv.glmnet` from the `glmnet` package.

In `Python` several `glmnet` "ports" are available to perform LASSO regularization, but check whether it runs on the OS you use, and whether it can handle both Poisson regression and logistic regression. As far as I can tell `pyglmnet`[1] should be able to do the job. I haven't been able to find a `Python` package that performs stepwise model selection for Poisson regression. It shouldn't be too difficult to build this on top of the `statsmodels` package though, and you are allowed to use gen AI to write the required code. All in all, it seems easier to work in `R` for this assignment.

[1] `https://github.com/glm-tools/pyglmnet`

References

[1] Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)*, pages 31 – 41. IEEE CS Press, 2010.