# Risk-Assessment Model

## Data Synthesis

**1.1 - Justification**

*~~Lack of public datasets; specific metrics chosen~~*

*~~Need to consider influence of our data-generation on the relationships learned by the model~~*

Early in the development, we decided that the predictive Risk-Assessment Model should be trained on an artificially-generated dataset of simulated projects. This decision was made due to two main reasons: first, real datasets of successful software projects are not readily available (or at least, we were unable to find such datasets in our research), and second, if such a dataset does exist, then it is unlikely to include the same metrics our model tracks, so it would be of limited use for training. However, the decision to artificially generate a dataset introduces a significant element of bias into our system, as the model may only learn the relationship(s) present in the dataset. As such, it is crucial that the generated data reflects real-world software development projects. Alternatively, if the project were of a larger scale or were allocated more time, then we could pursue obtaining our own dataset via surveys sent to project managers and software development companies.

**1.2 - Project Simulation**

*~~SimProject object - represents a single simulated project~~*

*~~- attributes initialised arbitrarily, development simulated with progress based on metrics and team experience.~~*

*~~- store a list of states during the development (similar to our actual HardMetric databases)~~*

*~~Unexpected Events (costs/neg. Progress)~~*

*~~Research: Developer Cost/Day (relative to rank)~~*

*~~Research: Commit Frequency -> Repo Activity~~*

*~~Research: Bug defect rate~~*

*~~If support for the project drops too low, it can be cancelled (ended prematurely)~~*

*~~-> all Success fields are set to 0 (Failure) to indicate the project was cancelled~~*

Sigmoid Function - used to map an arbitrarily large value to the range [0,1]

Exponential Functions - to avoid the uniform distribution provided by random.randint()

The main component of the simulation is the SimProject object which represents a single artificial project, whose state is sampled regularly throughout its progression. Initial hard metrics such as budget and deadline are randomly-generated within a given range, while soft metrics relating to the team and their experience are generated arbitrarily. As such, projects can be generated for a specific range of budgets/deadlines, but the remaining attributes may take any values, which provides variation in the dataset.

The *.simulate()* method emulates the lifetime of the project, with a new state (dataframe row) being recorded once per simulated day. Over the simulation, the progress of the project is maintained and incremented via a function on the team experience and soft metrics, with some randomness. In some cases, projects may experience “negative” progress, which is a probabilistic event that is more likely to occur in teams with low morale. Similarly, while the cost of development is regularly updated as it progresses, projects may also incur “unexpected costs” via a similar probabilistic mechanism. These features aim to replicate some of the challenges faced by projects, as development may not progress entirely as expected or as planned.

Within the simulation, a project may also be cancelled prematurely, with a probability inversely related to the managerial support. When a project exceeds its budget or deadline, the support metric begins to decline geometrically, thus increasing the chance of a cancellation. When a project is cancelled, it is considered to be a failure, as no viable product was produced.

There are several other simulated components of interest, having been influenced by relevant research. First, the main cost faced by the simulated projects is that of the development team (I.e. wages), which is modelled as a function of the experience rank of the developers. The relationship is shown in Table 1, but can be expressed as: .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rank** | **Experience** | **Hourly Wage** | **x Daily Hours** | **= Daily Wage** | **Calculated Daily Wage** |
| 1 | Beginner | £15-20 | 8 | £120-160 | £75 + £75x1 = £150 |
| 5 | Advanced | £40-60 | 8 | £320-480 | £75 + £75x5 = £450 |

**Table 1**: Developer Costs by Experience. Hourly Wages provided by <https://www.approvedindex.co.uk/software-developers/software-development-prices>

Next, from the code-base metrics, the simulation is only concerned with two areas in particular. First, the total number of code commits is simulated, as the change in commit frequency can be compared to estimate the relative activity of the development team. Moreover, our research found a study investigating the relationship between commit frequency and project success [**Source**: *https://www.academia.edu/36780849/The\_empirical\_commit\_frequency\_distribution\_of\_open\_source\_projects*] which measured project activity by the following formula:

Since the study only concerned Open-Source projects and our system is more suited to commercial, closed-source project, we decided to adapt the formula to consider the commit frequency for the last month (30 days) instead of 6 months. In particular, this metric is used to evaluate

The second modelled code metric is the rate at which known bugs are resolved via commits, in order to provide an estimation of the quality of the project’s code and the engagement of the development team with known issues. For each developer, the number of commits is generated probabilistically, based on developer experience and soft metrics such as the team commitment and morale; then for each commit, the developer has a probability of introducing a new bug or fixing an existing a bug. As the project progresses, the resolution rate tracks the ratio of resolved bugs over the total number of known bugs. One limitation of this approach is that defect rate may only provide a partial view of the code, as this metric’s accuracy depends on issues being identified and flagged [**Source**: *https://www.sealights.io/code-quality/defect-density-context-is-king/*].

**1.3 - Success Evaluation**

Potentially flawed approach

Use weighted sum of the five components

Each component has a minimum threshold required to be considered “successful”

Binarization of Success (0/1) so ML can learn irrespective of score

Propagate success values back to all previously recorded project states (produce labelled dataset)

Then sample fixed number (3-5) states per project.

Also, want the model to learn for multiple points in project development

-> sample each project more than once

-> need to find balance between undersampling and oversampling

**1.4 - Data Generation**

Written to CSV file - easy import to dataframe; keep separate from main DB

Originally, randomly spread training data throughout large range

Want model to support wide range of budgets (1-3M)/deadlines (1-1500 days)

*Problem*: data had poor coverage of the relevant cases, so the model would struggle to learn the problem-specific cases

Switch to batch-generation to yield better performance

-> project bdgt/dl more evenly distributed

-> However, need to ensure we don’t over-train for the problem by producing massive amounts of data; the model should learn/infer the relationship, rather than being shown all cases

-> balancing act between the granularity of the batches and the “random” spread

(RESEARCH) most projects have a cost of $100k-300k (source)

-> focus most data generation on the interval 100-500k, 1-500 days

-> then, generate fewer bigger budgets (500k-1.5M) and longer projects (500-1500)

Export to CSV for easy import to Pandas Dataframe

Randomly-initiated projects within interval

## Model Training

**2.1 - Model Structure**

Initially favoured using Logistic Regression for classification

Worked well, but Logreg doesn’t support partial\_fit.

Switched to using SGDClassifier

One model for each component (Timescale, Finance, Team, Code, Management) and another for the overall success.

Design choice: each model learns from all independent fields

-> to avoid focusing the model on certain columns and influencing the relationship

-> however, there is a risk of overfitting when training on many fields (SOURCE)

**2.2 - Avoiding Overfitting**

Train/Test Splits

Mean Squared Error (want to minimise)

Tuning

Crossvalidation - GroupKFold

(given multiple samples per project, need to ensure project only appears in either training or test, not both - prevent leakage of test data)

GridSearchCV to find C, regularisation term

**2.3 - Performance Evaluation & Export**

MSE most suited for regression, BUT we are using a Classifier

Precision, Recall, F1 - PR-Curve

Generate separate test data-set

(Limitation: uses same simulation, but can vary budget/deadline)

Exporting the Trained Model with Joblib rather than Pickle

(apparently faster for large models [CITE])

Also write-out model accuracy also to determine (rough) model likelihood of correctness

## Risk-Assessment Predictions

**3.1 - Model Confidence Probability + Bayes**

Is a chance the model is incorrect, so we consider that in the prediction

True Success Rate = Precision for class Success

False Failure Rate = false failure / (true failure + false failure)

For example, if the model with an 80% accuracy rate, predicts a

project will succeed with 68% confidence, then we also consider the 20% chance that the model is incorrect.

P(actual success) = 0.8 \* 0.68 + 0.2 \* 0.32

**3.2 - Risk Assessment Generator**

Only need a project’s datapoint to get a Risk-Assessment

## Evolution & Improvements

**4.1 - Relearning**

Need partial-fit;

To avoid overfitting to new data, only limited samples (3) are taken from each completed project

Train once a week, all-together

Success values obtained from final survey, not from evaluation formulae

## 4.2 - Improvements

Big Improvements possible in accuracy of data-generation

-> more fields; more complex relationships modelled (heat=Crunch Time)

-> could also continue to simulate projects after completion

Model could consider more granularity in assessment

E.g. scale of success (1-5) rather than binary