Summary

This solution is inspired by simple spam filtering algorithm. It uses neural network to find potential solutions and processes them to find the most appropriate one. I've not included any specific information about NN such as how many layers it should have because the algorithm hasn't been tested on any data yet.

Input

Let's say we have a dictionary which consists of the most common English words (in its simple form) and common words in programming world like 'python', 'apt-get' etc. (maybe it's worth to analyse training set to find such words).

Input is represented as a vector of the same length as a dictionary. The *i-th* element of the vector equals:

- 1 if the *i-th* word occurs in the given crash report
- 0 otherwise

Output

Let's say we have a set of all solutions used by our system (shell scripts for example). Output is represented as a vector of the same size as a set of solutions. The *i-th* element of the vector represents probability that *i-th* solution MAY be helpful. I assume that for one crash report there are several possible causes and therefore several possible solutions.

Algorithm

Given a crash report we use our model to get probabilities of solutions. We take those which have at least probability *p* for further consideration.

Next, let's say that every solution has conditions assigned to them and if any of these conditions is true then the solution cannot be used. Conditions should use informations about user's environment (installed packages, name of the app which returned error). For example if our model proposes 'apt-get install python' as one of the solutions and python is installed then this solution is rejected (condition saying 'python is installed' seems to be reasonable for solution 'apt-get install python').

Solution to be proposed is chosen from remaining ones and in my opinion it should be one which is the most successful (client app should collect information if solution was helpful) and in case of failure should try another solution.