Changes and Improvements

Exploring the Relationship between Design Metrics and Software Diagnosability using Machine Learning (2018)

Thomas Dornberger Sofie Kemper

2018-06-28

Since our team had already implemented a lot of features and generated a lot of data, our main challenge is the selection of meaningful predictors of software diagnosability. Hence, our insights described in the following mostly relate to how we pick features and try to optimise our machine learning strategy.

1 Changes made to the features

1.1 Features

We added a measure of dynamic complexity to supplement our information about static complexity. As suggested by Mojdeh, we also use a combination of these two features.

Our main challenge is the selection of relevant features. We "handpick" features to use for the model in a strategic way: We consider insights gained from correlation analysis as well as variance in the features (commonly suggested for machine learning problems) and previous results by other researchers. We build our models in the following: We start with one feature per linear regression model, evaluating all features that seem promising. Based on the scores obtained by these models, we choose the most promising feature(s) and build a model containing this feature as well as other promising features that are not highly correlated to the former feature. All models are evaluated using *mean squared error* to help us choose the objectively best features. Using this approach, we build our model by adding one feature at a time.

We have already seen that we need to consider combinations of features from different categories (e.g., static and dynamic features) in order to represent different characteristics of the data. It seems as though static, project-wide metrics are not as important for the software diagnosability as some other features. As we assumed, the quantity of features is far less important than the quality of features which is why we use the feature selection approach explained above.

1.2 Targets

We have made some changes to the targets. We consider two values for each operationalisation of diagnosability (DStar, Tarantula, etc.): First, we consider the rank of the score of the faulty method – using the most suspicious faulty method, i.e., the one with the highest suspiciousness score, in case there are several. In addition we consider the "wasted effort" or the number of components a developer has to examine until he finds the (first) faulty method. For the latter we adapted an average-case method rather than a worst-case method as suggested by Mojdeh. This means that in case there are several components with the same rank as the real faulty method, we assume that the developer will look at half of them (in addition to all components with a lower rank) until he finds the real fault instead of having to look at all of them as considered before. We believe that this is a more accurate representation of reality.

We have noticed that DStar cannot be calculated for the true faulty component in a few cases, for instance org.jfree.chart.block.RectangleConstraint#89 in Chart version 13: Since there is only one test case that fails and that is also the only one that covers this method, the denominator in the DStar formula is 0 and, thus, the metric cannot be calculated. This problem only occurs very seldomly but still needs to be handled. We have decided to treat this non-calculable score as the worst possible performance of the fault-localisation technique: The rank of these elements' suspiciousness is the maximum rank present plus one and we assume that the developer has to examine every single component that was given a score first before looking at the elements for which a score could not be calculated. We consider this perspective the most realistic way of dealing with this problem since it mirrors the order in which a tool would suggest suspicious components to the developer. Although a tool might calculate different scores and base its recommendations on that to avoid problems such as non-calculable DStar scores, our fault localisation approach is limited to applying one algorithm at a time to ease comparability as well as implementation complexity. However, we still believe that it should reflect a "real world" perspective quite well.

We realised that the exponent in the DStar formula is not important in our case since exponentiation of positive values is monotonic, i.e., doesn't change the ordering of values and we are only interested in the suspiciousness score of a component *relative* to the other suspiciousness scores not the absolute values. Although we have considered using absolute values to compare the results to the ones obtained using our approach, the latter seems more representative of real usage of fault localisation to us as explained above.

2 Changes made to the methodology

We considered trying classification in addition to our linear regression but do not think this is a good idea unless we obtain absolutely no relevant results using linear regression. Our reasoning behind this choice is that classification will always aggregate the data and, thus, we will lose some of the information we have. Using a threshold for classifying diagnosability into distinct categories always endangers the results' validity since any threshold we could choose is in some way arbitrary.

When using a subset of the available data, we saw that principal component analysis (PCA) works very well as a dimensionality reduction technique for the data: We could reduce the data's dimensionality by more than 90% and still explain 90% of the variance in the data via the principal components. As we have also seen when performing correlation analyses, many features seem to reflect similar tendencies in the data, i.e., are highly correlated. However, there were some problems when trying to apply linear regression to the transformed data: The results we obtained were a lot worse than the ones using "handpicked" features. We will have to reevaluate these results using the complete dataset but if the problems persist we will have to use only "handpicking" of features as dimensionality technique. It could be possible that the features which display the most variance in our case are not important factors in predicting diagnosability. Although this is contrary to common machine-learning wisdom, it is a possibility that we have to consider and deal with in a suitable manner, e.g., as explained above.

For PCA, we use a normalisation of our data to ensure that all features are scaled to zero mean and unit variance. This is a standard procedure in machine learning since it helps deal with data that might have been measured on different scales or in different domains. It is not important for linear regression (since the coefficients generated using linear regression) already scale the data as needed, but it is helpful for PCA.

For all our methods, we have introduced a train-test split to better evaluate our results. It is possible to apply linear regression without a test set by using values such as the *r-squared* (or *adjusted r-squared*) values as measures of quality. Optimising models based on these parameters guarantees that the models fit the data well. However, we do not know whether models obtained in such a way would actually be generalisable. Hence, we use 20% of our dataset (randomly chosen in a reproducible way) as test set and evaluate the obtained models with it using *mean squared error* as loss function. With this methodology, we try to ensure that our results are as relevant and generalisable as possible.

There are several other approaches that we considered and deemed interesting. However, due to time constraints we chose to pursue the more promising methods explained above. If we have the time, the following are two other machine learning approaches, we would like to try.

Applying decision trees might not produce a very accurate model as end result, but some insights could be gained by the upper-level splits. Decision trees usually apply that split (out of all possible feature-value combinations that could be used as split) which separates the data best, i.e., which produces the most homogeneous or "pure" split of the data into subsets regarding the target variable. Using the fault localisation accuracy (as defined above) as target value, this procedure could help us gain insight into which features are especially helpful in predicting software diagnosability. It might be particularly interesting if there are any concrete thresholds which affect diagnosability. Although decision trees are mostly applied to categorical target values, it is possible to use them for continuous variables. This is already implemented in many standard machine learning packages for R and python.

Decision trees seem like a promising method to study since they deal with high-dimensional data very well: due to the algorithm that is applied, the most important

features are chosen and less informative features are ignored. The resulting insights could be interesting research results and could be used to improve other models and feature selection.

Another interesting idea might be some form of unsupervised learning. This could help us gain a deeper understanding of the data and whether there exist general any characteristics that naturally divide the data into different classes. One approach would be applying clustering to the data. The resulting clusters could be examined based on our target values, e.g., by looking at the mean diagnosability score in each cluster as well as the per-cluster variance of this score. In case these scores significantly differ between different clusters, we would have found a link between this natural classification of the data and our notion of software diagnosability.

However, there are several caveats: Clustering is very difficult using our data due to the high number of dimension and low amount of available samples. We would need to significantly reduce the data's dimension before being able to apply standard clustering algorithms, thus, leading back to the feature selection problem that we encounter for other methods of machine learning. In addition, we have no idea how many clusters we would expect in the data, which makes certain clustering algorithms such as k-means difficult to apply. Lastly, we have no idea whether the "natural classes" according to our selected features in our data would be in any way indicative of software diagnosability. Hence, this approach seems like a lot of effort for less promising results.