Empir Software Eng (2012) 17:62–74
DOI 10.1007/s10664-011-9182-8

On the dataset shift problem in software engineering prediction models

Burak Turhan

Published online: 12 October 2011
© Springer Science+Business Media, LLC 2011
Editors: Martin Shepperd and Tim Menzies

Abstract A core assumption of any prediction model is that test data distribution does not differ from training data distribution. Prediction models used in software engineering are no exception. In reality, this assumption can be violated in many ways resulting in inconsistent and non-transferrable observations across different cases. The goal of this paper is to explain the phenomena of conclusion instability through the *dataset shift* concept from software effort and fault prediction perspective. Different types of dataset shift are explained with examples from software engineering, and techniques for addressing associated problems are discussed. While dataset shifts in the form of sample selection bias and imbalanced data are well-known in software engineering research, understanding other types is relevant for possible interpretations of the non-transferable results across different sites and studies. Software engineering community should be aware of and account for the dataset shift related issues when evaluating the validity of research outcomes.

Keywords Dataset shift · Prediction models · Effort estimation · Defect prediction

1 Introduction

Software engineering community has widely adopted the use of prediction models for estimating, e.g. development and maintenance cost/effort, fault count/density, and reliability of software projects. With the goal of building generalized prediction models in software development context, the community is faced with the challenge of non-transferable results across different projects/studies. In order to determine whether individual results are indeed transferrable across different sites, first the

B. Turhan (✉)
Department of Information Processing Science, University of Oulu,
POB.3000, 90014, Oulu, Finland
e-mail: turhanb@computer.org

validity of the assumptions yielding these results should be examined for understanding their effects on the results. This paper aims to introduce the *dataset shift* concept (Candela et al. 2009; Hand 2006), to interpret the variability of results across different sites from a data oriented perspective in addition to critiques of varying performance assessment techniques (Shepperd and Kadoda 2001).

Prediction systems aim to make generalizations from past experiences for estimating desired properties of future events. Such systems commonly operate under the assumption that future events to be estimated will be near identical to past events. More formally, a prediction system utilizes past (i.e. training) covariate-response pairs $\{x_{\text{train}}, y_{\text{train}}\}$ sampled from a joint distribution of the form $p(X, Y) = p(Y|X)p(X)$ with true conditional model $p(Y|X)$ and prior $p(X)$, for learning an estimated conditional $\hat{p}(Y|X)$ (i.e. the prediction model), in order to make predictions for future response variables given future (i.e. test) covariates, $\hat{p}(y_{\text{test}}|x_{\text{test}})$.

A prediction model is considered good to the extent that the estimated model $\hat{p}(Y|X)$ is a close approximation to the underlying true model $p(Y|X)$. This may be evaluated by various error criteria through many procedures by measuring the delta between the true and estimated model responses, and as the CFP for this special issue suggests, these may be causing the conclusion instability problem. Though non-standardized evaluation measures and procedures may well be one cause, this paper will assume an ideal state where the evaluation is not an issue, and rather focus on the other possible causes of the conclusion instability problem, namely data related issues. Specifically, any factor (i.e. $p(X)$, $p(Y)$ or other significant confounding factors) that affects the joint density $p(X, Y)$, and is changing between training and test environments will also affect the performance of the prediction model $\hat{p}(Y|X)$, unless accounted for during modeling.

The rest of this paper elaborates on the different types of dataset shift and their implications to software engineering field, provide pointers to computational methods to deal with certain types of dataset shift, and concludes with offering explanations by matching dataset shift types to certain reported (in)consistencies in software engineering literature.

2 Types of Dataset Shift

The problem of dataset shift has recently attracted the attention of noted researchers (Candela et al. 2009; Hand 2006). In particular, Storkey classifies different forms of dataset shift into six groups which are discussed in the following sections (Storkey 2009). Some groups are well-known such as sample selection bias and imbalanced data, while others may be relevant, but are not appropriately or explicitly addressed in, software engineering research results. The descriptions of different dataset shift types are based on Storkey (2009) and are interpreted from software engineering point of view. Note that the different types of dataset shift are not mutually exclusive, on the contrary may be related in many cases.

2.1 Covariate Shift

Covariate shift occurs when the distributions of covariates in training and test data differ, i.e. $p(X_{\text{train}}) \neq p(X_{\text{test}})$. In this case, if the prediction model $\hat{p}(Y|X)$ is able to correctly approximate the true model $p(Y|X)$ globally, i.e. at least for $p(X_{\text{test}})$ or $p(X)$ in general, then there are no implications. Otherwise, the prediction model might be a good approximation to the true model in the X-space only for the locality defined by $p(X_{\text{train}})$, but not necessarily for $p(X_{\text{test}})$. For instance, the true global model can be a high degree polynomial and can be successfully approximated with a linear prediction model with a positive-slope for the region of X-space covered by training samples, whereas the region covered by test samples may show linear characteristics with a negative slope, i.e. a different local model.

Implications for software engineering domain could be for the typical size based prediction models. Size is a commonly used covariate for effort or fault estimation, and corresponding models might be effective for project sizes within the traditional operational values of a company. Due to new business opportunities, or change in technologies and development techniques, the size of new projects might differ from the past.[1] Such conditions require re-examination of existing prediction models for covariate shift. Using data from projects of different scales and types, within or across companies (this will also be addressed in Section 2.6), also requires control for covariate shift. For instance, COCOMO incorporates local calibration for handling this issue (Boehm et al. 2000).

Another important implication is for the design of simulation studies, where training and test set splits, with or without cross validation, might result in covariate shift among resulting datasets, and should be compensated for. Covariate shift problem is an active topic in machine learning community, and proposed solutions range from importance weighting of samples to kernel based methods (Bickel et al. 2009; Huang et al. 2006; Sugiyama et al. 2008), which will be discussed in Section 3.

2.2 Prior Probability Shift

Prior probability shift occurs when the predictive model is obtained via the application of Bayes rule, $p(Y|X) = p(X|Y)p(Y)$, and the distributions of the response variable differ in training and test data, i.e. $p(Y_{\text{train}}) \neq p(Y_{\text{test}})$. In this case the prediction model fails to properly approximate the true model. Note that the direction of causal relation among covariates and response variables change in prior probability shift, i.e. covariates are dependent on the response, whereas it is the opposite for covariate shift (Storkey 2009).

In a software engineering context, prior probability shift may occur, for instance, in fault prediction studies when the fault characteristics change in new projects as a result of process improvement for better development, testing and quality assurance activities that are not captured by the covariates, or simply due to different characteristics of the test project. Furthermore, specialization in a business domain and increased developer experience over time, and changes in business domain may similarly affect the fault characteristics. To use fault models across projects with different fault characteristics require compensation for prior probability shift. It is easy to augment the model accordingly by simply accounting for the shifted distribution $p(Y_{\text{test}})$, if the new distribution is known. While this can be simulated

[1]For simplicity, it's assumed that there are no other confounding effects of such changes than on software size.

in empirical studies,[2] where this is known a priori, in practice this information will not be available. Therefore, $p(Y_{\text{test}})$ should be parametrized and computed by marginalizing over the parameter space and training covariates (Storkey 2009).

2.3 Sample Selection Bias

Sample selection bias is a statistical issue, and for the purposes of this paper a dataset shift type, that is well-known and accounted for in software engineering studies, and listed here for the sake of completeness. In software prediction studies, it should be considered that companies collecting relevant data have mostly higher maturity levels than general software engineering industry and selected projects may not reflect the usual operating environment of a company, hence conclusions from such data may not be externally valid. Increasing the scale of these kind of studies to address this validity problem, however, eventually surrenders to the paradox of empirical research that it is practically impossible to draw meaningful conclusions from the resulting population which becomes too general to characterize (also see Section 2.6). Nevertheless, in the micro level, sample selection bias might be introduced in a controlled manner to deal with (1) covariate shift through relevancy filtering in Kocaguneli et al. (2010) and Turhan et al. (2009), (2) imbalanced data through sampling techniques (Menzies et al. 2008), (3) and source component shift through logical grouping of data (Bakır et al. 2010; Premraj and Zimmermann 2007).

2.4 Imbalanced Data

Imbalanced data is concerned with the cases where certain type(s) of events of interest are observed rarely compared to other event types. This is a common issue in software fault prediction studies, where the number of non-faulty entities dominates the data sample as opposed to the number of to faulty entities. Since the goal is to learn theories about this rare event (i.e. faults), a widely accepted solution is to introduce a sample selection bias on purpose via the application of over/under sampling techniques. In practice, this causes a prior probability shift when the learned model is applied in the test settings, and it should be addressed with an adjustment of the estimates of the model as explained in Section 2.2. While Storkey argues that failure to compensate may be seen as a way to address the relative importance of false positives and true positives (Storkey 2009), it gives no control to the practitioner for adjusting this relative importance. In the worst case, if the losses associated with false positives and true positives are equivalent, or if a specific loss function is desired, then the fault prediction models will not be optimized with respect to the preferred criteria, unless the prior probability shift is addressed.

[2]However, this would introduce an unfair bias, since it would mean using, during model construction phase, information related to an attribute that is to be predicted (i.e. defect rate). The model is supposed to predict that attribute in the first place, and should be blind to such prior facts that exist in the test data.

2.5 Domain Shift

Domain shift refers to cases where a concept can be measured in different ways, and the method of measurement changes between training and test conditions. Immediate relevance to software engineering is again size based models. Size is a concept and can be measured in different ways from code or earlier in terms of function points (FP). For instance, different values (with different units) are obtained with the application of different methods for the latter (Demirors and Gencel 2009), and similar problems arise in counting the lines of code (LOC). However, such size measures are usually reported only as LOC or FP without providing measurement details. Before applying a size-based model, i.e. to a new project, it must be assured that the training and test data are collected in a consistent manner. Otherwise, domain shift may lead to inconsistencies across studies for similar reasons as using different evaluation measures for performance (Shepperd and Kadoda 2001).

2.6 Source Component Shift

Source component shift take place when each, or groups of, datum in the data sample originate from different sources with unique characteristics and in varying proportions (Storkey 2009). This description is almost a perfect characterization of software engineering datasets, where data from several projects or companies with different properties are typically merged (Boehm et al. 2000; Lokan et al. 2001). In this type of shift, due to their specific characteristics, each source component might span a different region of the problem space, leading to too generic models. Experimental procedures, i.e. cross validation or train/test splits, would probably cause covariate shift as well. Furthermore, as the proportion of sources may vary among training and test conditions, prior probability shifts both in the ratio of sources and response variables are inevitable. In software engineering, specifically in cross-project prediction, studies *source component shift* has been referred to as data heterogeneity (or homogeneity). The main idea of such studies has been to process datasets (e.g. repartitioning through relevancy filtering, analysis of variance, or actual data source) to achieve more homogeneous structures (i.e. identifying the source components) to work with Briand et al. (2002), Kitchenham et al. (2007), Turhan et al. (2009), Wieczorek and Ruhe (2002), and Zimmermann et al. (2009).

3 Managing Dataset Shift

This section aims to answer the question that, by now, should have arisen in your mind: "What needs to be done?". There is no silver bullet, but there exist certain techniques to address different types of dataset shift to some extent.[3] In this section, some representative techniques are discussed in two broad groups, namely instance-based and distribution-based techniques.

[3]Domain shift is not included in the discussion, since that is a measurement related issue that should be separately handled by the researcher/practitioner.

3.1 Instance Based Techniques

3.1.1 Outlier Detection

Outliers are data points that are not of interest from statistical modeling point of view, but have the potential to significantly affect the dataset structure and model parameters (Chandola et al. 2009). In other words, *outliers may cause (covariate) shifts in datasets* and consequently lead to false generalizations. While the choice to remove or keep outliers in data analysis is a topic of discussion, it is paramount to be aware of their existence, to make necessary assessments and to take required actions accordingly.

In this respect, studies reporting mean and standard deviation values for descriptive statistics and performance results would be misleading if outliers exist and are not removed, since these statistics are sensitive to outliers. Median and quartile values should be preferred in such cases as they are more robust to outliers than mean and standard deviation. Further, the robustness of different models against outliers may vary, therefore, software engineering studies that compare different models to select the "best" alternative should take this into account. In an hypothetical scenario, the observed superiority of a complex model A against a simpler model B on a given dataset may be purely due to model A's robustness against outliers, and model B may perform as good as model A when outliers are removed. Hence it should be noted that while it is tempting to use readily available tools (i.e. WEKA (Hall et al. 2009)) for constructing predictive models, it is important to be aware of the limitations and strengths of the underlying models rather than treating them as black-boxes.

In some cases outliers themselves may be the objects of interest in prediction. Therefore, it is important to differentiate outlier detection from outlier removal. Analysis of outliers could reveal the limitations of (a class of) predictive models. Further, the context plays an important role in defining the outliers. For instance, it does not have any practical value to construct a defect prediction model with an exceptional goodness of fit, if the removed outliers account for an undiscardable number of defect logs as well. This would yield the wrong impression that the predictive model is able to detect almost all defects. In fact the model would be limited to detect certain types of defects and it would be missing a significant number of them. Hence, the initial goal, which is predicting defects in this case, should neither be forgotten nor sacrificed for statistical excellence.

Though most software engineering studies tend to discard reporting the issue, outlier detection has been addressed explicitly in a number of software engineering publications. For instance, Briand et al. emphasize the importance of identifying (and removing) both univariate and multivariate outliers, and advises using Pregibon beta for logistic regression, Cook's distance for ordinary least squares and scatter plots in general (Briand and Wust 2002). Further, Kitchenham et al. employ jackknifed cross-validation to achieve robustness against outliers (Kitchenham et al. 2007), and Keung et al. developed a methodology, named Analogy-X, to identify outliers (Keung et al. 2008). Boxplots are also commonly used in software engineering studies for visual inspection of datasets and results (Menzies et al. 2008; Turhan et al. 2009). Another useful technique for visualizing and detecting outliers in time-series data is VizTrees by Lin et al. (2004). It is out of the scope of this paper to give details on the well-founded field of outlier detection. We refer the reader to the extensive survey by

Chandola et al., where they group and discuss specific methods under classification-based, nearest-neighbor-based, clustering-based and statistical techniques (Chandola et al. 2009).

3.1.2 Relevancy Filtering

There is no agreed upon definition of an outlier; its definition is rather context dependent (Chandola et al. 2009). Relevancy filtering exploits this fact to introduce a controlled sample selection bias that is tweaked towards the test set samples. Specifically, relevancy filtering allows the use of only certain train set samples, which are closer to the test set samples based on a similarity/distance metric, for model construction. Please note that only the covariates of test data should be used in calculating the similarity metric, and not the responses.[4] An important implication of using relevancy filtering is that there is not a global static model, but rather a new model is dynamically constructed based on different subsets of training data each time a new test instance/batch arrives. This may be considered as a limitation of relevancy filtering that it requires the underlying model to be simple enough to enable on the fly construction for performance issues.

Relevancy filtering addresses *covariate shift* by forcing a training distribution that matches the test distribution as well as possible. However, there is a risk of introducing a prior probability shift since the distribution of responses in the training set cannot be guaranteed to be preserved. Nevertheless, relevancy filtering yielded promising results in software fault prediction (Turhan et al. 2009) and cost estimation (Kocaguneli et al. 2010; Kocaguneli and Menzies 2011) studies for selection of relevant training data to construct prediction modes. For example, Turhan et al. was able to significantly improve the performance of naive Bayes prediction models across different projects (i.e. learn from a set of projects then apply to another project) using nearest-neighbor similarity (Turhan et al. 2009).

3.1.3 Instance Weighting

While relevancy filtering can be considered as hard-filtering (i.e. an instance is either in or out), instance weighting is a soft-filtering method that assigns a weight to each training instance based on its relative importance with respect to the test set distribution. Similar to relevancy filtering, instance weighting specifically addresses *covariate shift* problem. In this technique, determining the weights becomes the essential issue. Once the weights are set, it is possible to use the weighted versions of commonly used models (e.g. weighted least squares regression, weighted naive Bayes (Zhang and Sheng 2004)).

Recent progress in machine learning research led to the development of techniques for accurate weight identification. Shimodaira proposes to identify weights based on the ratio of training data distribution to test set distribution (Shimodaira 2000). Figure 1 (from Shimodaira 2000), shows the effectiveness of the approach on a toy example with true data generated by adding white noise to a third degree

[4]In practice, this warning applies to simulation studies, since test responses are typically not known in real settings.

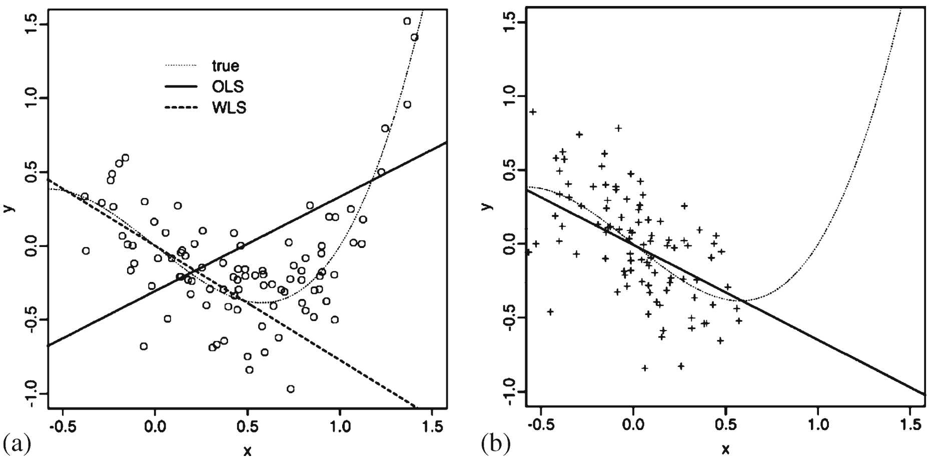

Fig. 1 Example of instance weighting for comparing WLS and OLS performance in the existence of covariate shift, from Shimodaira (2000)

polynomial. The plot on the left shows the fit of final models (weighted least squares (WLS) vs. ordinary least squares (OLS)) on the training set and the plot on the right shows the fit of WLS on the test set. Please note the covariate shift in the test set and how it is successfully handled by WLS. It may also be argued that the region of input space that contributes data points to the training set, but not to the test set, contains outliers. In both situations, instance weighting overcomes covariate shift due to either change in the distribution or outliers.

Another method for instance weighting, KLIEP, has been recently proposed by Sugiyama et al. which minimizes the Kullback–Leibler divergence between training and test set densities using kernel-based methods without explicit density estimation (Sugiyama et al. 2008). Sugiyama et al. argues that KLIEP outperforms other techniques such as Bickel et al.'s formalization as an optimization problem, and kernel mean matching (KMM) proposed by Huang et al. (2006). Please refer to relevant publications for more details.

3.2 Distribution Based Methods

3.2.1 Stratification

In simulation based settings stratification, or stratified sampling, ensures that *prior probability shift* does not occur. Such studies using cross validation or training-test data splits should prefer stratification over random splits in order to preserve the ratio of different response groups in the newly created data bins. In classification problems such as fault prediction, its application is straightforward. In regression type fault prediction and effort estimation studies, possible solutions are to devise logical groups (i.e. five categories of effort ranging from very-low to very-high) based on pre-determined thresholds or to identify some number of clusters of response variable values, and then to reflect the ratio of these logical groups/clusters in the whole dataset into the training and test sets.

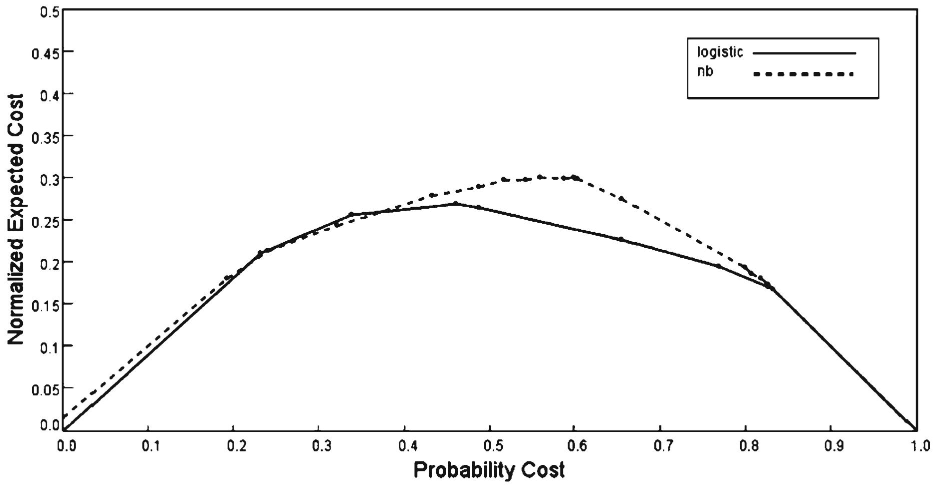

Fig. 2 Example cost curves for comparing two fault prediction models, from Jiang et al. (2008a)

Stratification is also helpful in cases of *imbalanced data*. In this case, random splits may result in the exclusion of the minority instances from most of the newly created bins, which would cause numeric errors in performance metric calculation and the corresponding bin-models to be meaningless.

However, it should be noted that while stratification avoids prior probability shift in simulation settings, it also assumes the real world data will not suffer from prior probability shift after the model is deployed. Therefore, it is useful to utilize stratification in simulation settings for post hoc analysis, but this does not solve the problems associated with prior probability shift between simulation and real-settings. This problem is addressed with cost curves which are explained next.

3.2.2 Cost Curves

In addition to the problem stated above, the problems associated with the application of over/under sampling to *imbalanced data* was discussed in Section 2.4, arguing that sampling strategies cause prior probability shift and affect the loss functions in an uncontrolled manner. Sampling techniques may be preferred, and are commonly used, over stratification as a design decision in predictive model construction, yet both approaches are prone to *prior probability shift and changing relative costs* in real settings.

For binary classification problems (i.e. fault prediction), Drummond and Holte's cost curves provide control over all these complications at once (Drummond and Holte 2006). Cost curves visualize predictor model performances "over the full range of possible class distributions and misclassification costs" for binary classification problems (Drummond and Holte 2006). Cost curves provide decision support by visual inspection of all possible future scenarios for class ratios and loss functions. Therefore, it is advised to include cost curve analysis in empirical studies of predictive models in order to see the capabilities of reported models over the space of all possible future states.

Cost curves are investigated by Jiang et al. in software engineering domain along with a comparison with alternative methods for model selection including lift charts and ROC curves (Jiang et al. 2008a). An example of cost curves is shown in Fig. 2 (from Jiang et al. 2008a) that compares the fault prediction performances of logistic regression and naive Bayes classifiers on KC4 dataset from NASA MDP repository. In order to determine the cost curve for a model, the points in a ROC curve are converted to lines and the convex hull, including the x-axis, with the minimum area is found. In the example, logistic regression turns out to be a better model, since its curve is consistently closer to the optimal cost curve (i.e. x-axis) for all scenarios. Please refer to Drummond and Holte (2006) for more details on cost curves and to Jiang et al. (2008a, b) for its applications in software fault prediction.

3.2.3 Mixture Models

In order to address source component shift, where data are known to have different origins, individual source components in the data should be accounted for. These components can be either identified manually when there is information about the origin of data, or estimated and handled automatically with mixture models such as "mixture of Gaussians" and "mixture of experts" (Alpaydin 2010; Storkey 2009).

In manual identification, meta-knowledge can be used for defining alternative source components, e.g. company, domain, project team, or individual team members. As an example of manual identification Wieczorek and Ruhe compared company-specific vs. multi-company data for cost estimation using the Laturi database consisting of 206 projects from 26 different companies, i.e. source components are defined at the company level (Wieczorek and Ruhe 2002). While they did not find any significant advantage of using company-specific data, the systematic review by Kitchenham et al. revealed that some companies may achieve better cost estimations with company-specific data (Kitchenham et al. 2007). Considering the different ways of defining source components, Wieczorek and Ruhe recommends domain clustering as an alternative, and Bakir et al. reports such an application in embedded systems domain (Bakır et al. 2010).

When it is not possible to identify the number of source components for the data, the latter approach—automated mixture models—can be used instead. Mixture models address multi-modalities (i.e. different source components) in the data as opposed to the uni-modality assumption of their counterparts. Therefore, the idea of mixture models is to identify multiple density distributions (commonly from the same family, e.g. Gaussian) and to fit different models for each density component. In practice, a test datum is assigned to a single source component and a prediction is achieved based on the specified model for that component. As an alternative, an aggregation of all predictions can be taken into account based on a weighting of the posterior probabilities that the test datum belongs to a certain source component. For an application of mixture models to fault prediction please see Guo and Lyu (2000).

4 Summary

In order to provide a possible explanation for the conclusion instability problem, this paper introduced the dataset shift concept, discussed its implications for predictive model construction for software engineering problems, and provided pointers to representative techniques to address associated issues.

Dataset shift offers viable justifications for certain results in software engineering:

- Covariate shift might be the cause for the inconsistencies reported by Shepperd and Kadoda across different training and test set splits, and seemingly better performance of case-based reasoning (Shepperd and Kadoda 2001), as well as Myrtveit et al.'s leave-one-out cross validation related issues (Myrtveit et al. 2005).
- Covariate shift also explains why COCOMO with local calibration is consistently selected among the best models in Menzies et al.'s extensive effort estimation experiments, along with source component shift as they logically grouped their data (Menzies et al. 2010).
- The benefits of relevancy filtering in Kocaguneli et al.'s effort estimation (Kocaguneli et al. 2010) and Turhan et al.'s fault prediction (Turhan et al. 2009) studies can be attributed to compensation for covariate shift through a controlled sample selection bias with relevancy filtering.
- Source component shift, together with covariate shift is referred to as *data heterogeneity/homogeneity* and addressed to some extent in certain software engineering studies (Briand et al. 2002; Kitchenham et al. 2007; Turhan et al. 2009; Wieczorek and Ruhe 2002; Zimmermann et al. 2009)
- The contradicting results in cross project/company studies can be explained either with covariate shift, prior probabilty shift, source component shift, or a combination of those (Kitchenham et al. 2007; Turhan et al. 2009; Zimmermann et al. 2009).

Dataset shift is a recently coined and active research topic in machine learning community, and the classification of dataset shift types and techniques described in this paper are not necessarily complete. Nevertheless, dataset shift provides a means to study the conclusion instability problem in software engineering predictions, and the community can benefit by validating and interpreting their results from this perspective.

Acknowledgements This research is partly funded by the Finnish Funding Agency for Technology and Innovation (TEKES) under Cloud Software Program. The author would like to thank the anonymous reviewers for their suggestions which greatly improved the paper.

References

Alpaydin E (2010) Introduction to machine learning, 2nd edn. The MIT Press, Cambridge, MA

Bakır A, Turhan B, Bener A (2010) A new perspective on data homogeneity in software cost estimation: a study in the embedded systems domain. Softw Qual J 18(1):57–80

Bickel S, Brückner M, Scheffer T (2009) Discriminative learning under covariate shift. J Mach Learn Res 10:2137–2155

Boehm B, Horowitz E, Madachy R, Reifer D, Clark BK, Steece B, Brown AW, Chulani S, Abts C (2000) Software cost estimation with Cocomo II. Prentice Hall, Englewood Cliffs, NJ

Briand L, Wust J (2002) Empirical studies of quality models in object-oriented systems. Adv Comput 56:97–166

Briand LC, Melo WL, Wust J (2002) Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Trans Softw Eng 28:706–720

Candela JQ, Sugiyama M, Schwaighofer A, Lawrence ND (eds) (2009) Dataset shift in machine learning. The MIT Press, Cambridge, MA

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. ACM Comput Surv 41(3):15:1–15:58

Demirors O, Gencel C (2009) Conceptual association of functional size measurement methods. IEEE Softw 26(3):71–78

Drummond C, Holte RC (2006) Cost curves: an improved method for visualizing classifier performance. Mach Learn 65(1):95–130

Guo P, Lyu MR (2000) Software quality prediction using mixture models with EM algorithm. In: Proceedings of the the first Asia-Pacific conference on quality software (APAQS'00). IEEE Computer Society, Washington, DC, USA, pp 69–78

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD explorations, vol 11/1

Hand DJ (2006) Classifier technology and the illusion of progress. Stat Sci 21(1):1–15

Huang J, Smola AJ, Gretton A, Borgwardt KM, Schšlkopf B (2006) Correcting sample selection bias by unlabeled data. Neural Information Processing Systems, pp 601–608

Jiang Y, Cukic B, Ma Y (2008a) Techniques for evaluating fault prediction models. Empir Soft Eng 13(5):561–595

Jiang Y, Cukic B, Menzies T (2008b) Cost curve evaluation of fault prediction models. In: Proceedings of the 19th int'l symposium on software reliability engineering (ISSRE 2008), Redmond, WA, pp 197–206

Keung JW, Kitchenham BA, Jeffery DR (2008) Analogy-X: providing statistical inference to analogy-based software cost estimation. IEEE Trans Softw Eng 34(4):471–484

Kitchenham BA, Mendes E, Travassos GH (2007) Cross versus within-company cost estimation studies: a systematic review. IEEE Trans Softw Eng 33(5):316–329

Kocaguneli E, Menzies T (2011) How to find relevant data for effort estimation? In: Proceedings of the 5th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM'11)

Kocaguneli E, Gay G, Menzies T, Yang Y, Keung JW (2010) When to use data from other projects for effort estimation. In: Proceedings of the IEEE/ACM international conference on automated software engineering (ASE '10). ACM, New York, pp 321–324

Lin J, Keogh E, Lonardi S, Lankford J, Nystrom DM (2004) Visually mining and monitoring massive time series. In: Proceedings of 10th ACM SIGKDD international conference on knowledge and data mining. ACM Press, pp 460–469

Lokan C, Wright T, Hill PR, Stringer M (2001) Organizational benchmarking using the isbsg data repository. IEEE Softw 18:26–32

Menzies T, Jalali O, Hihn J, Baker D, Lum K (2010) Stable rankings for different effort models. Autom Softw Eng 17(4):409–437

Menzies T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y (2008) Implications of ceiling effects in defect predictors. In: Proceedings of the 4th international workshop on predictor models in software engineering (PROMISE '08). ACM, New York, pp 47–54

Myrtveit I, Stensrud E, Shepperd M (2005) Reliability and validity in comparative studies of software prediction models. IEEE Trans Softw Eng 31(5):380–391

Premraj R, Zimmermann T (2007) Building software cost estimation models using homogenous data. In: Proceedings of the first international symposium on empirical software engineering and measurement (ESEM '07). IEEE Computer Society, Washington, DC, USA, pp 393–400

Shepperd M, Kadoda G (2001) Comparing software prediction techniques using simulation. IEEE Trans Softw Eng 27(11):1014–1022

Shimodaira H (2000) Improving predictive inference under covariate shift by weighting the log-likelihood function. J Stat Plan Inference 90(2):227–244

Storkey A (2009) When training and test sets are different: characterizing learning transfer. In: Quionero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (eds) Dataset shift in machine learning, chapter 1. The MIT Press, Cambridge, MA, pp 3–28

Sugiyama M, Suzuki T, Nakajima S, Kashima H, von Bünau P, Kawanabe M (2008) Direct importance estimation for covariate shift adaptation. Ann Inst Stat Math 60(4):699–746

Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. Empir Softw Eng 14(5):540–578

Wieczorek I, Ruhe M (2002) How valuable is company-specific data compared to multi-company data for software cost estimation? In: Proceedings of the 8th international symposium on software metrics (METRICS '02). IEEE Computer Society, Washington, DC, USA, p 237

Zhang H, Sheng S (2004) Learning weighted naive Bayes with accurate ranking. In: Proceedings of the 4th IEEE international conference on data mining, pp 567–570

Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction. In: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. ACM

Burak Turhan holds a PhD in Computer Engineering from Bogazici University, Turkey. He is a postdoctoral researcher in the Department of Information Processing Science at the University of Oulu, Finland. His research interests include empirical studies of software engineering, on software quality, defect prediction, cost estimation, as well as data mining for software engineering and agile/lean software development with a special focus on test-driven development. He is a member of ACM, IEEE, and IEEE Computer Society.