Maximum Profit Mining and Its Application in Software Development

Charles X. Ling[1], Victor S. Sheng[1], Tilmann Bruckhaus[2], Nazim H. Madhavji[1]
[1]Department of Computer Science, The University of Western Ontario
London, Ontario N6A 5B7, Canada
{cling, ssheng, madhavji}@csd.uwo.ca
[2] Numetrics Management Systems
20863 Stevens Creek Blvd, Suite 510, Cupertino, CA
tilmannb@numetrics.com

ABSTRACT

While most software defects (i.e., bugs) are corrected and tested as part of the lengthy software development cycle, enterprise software vendors often have to release software products before all reported defects are corrected, due to deadlines and limited resources. A small number of these defects will be escalated by customers and they must be resolved immediately by the software vendors at a very high cost. In this paper, we develop an Escalation Prediction (EP) system that mines historic defect report data and predict the escalation risk of the defects for maximum net profit. More specifically, we first describe a simple and general framework to convert the maximum net profit problem to cost-sensitive learning. We then apply and compare several well-known cost-sensitive learning approaches for EP. Our experiments suggest that the cost-sensitive decision tree is the best method for producing the highest positive net profit and comprehensible results. The EP system has been deployed successfully in the product group of an enterprise software vendor.

(Note: *this paper is accepted by KDD'06*)

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Management, Performance, Economics

Keywords

Escalation prediction, cost-sensitive learning, data mining

1. INTRODUCTION

Building large enterprise software is generally a highly complex and lengthy process, during which numerous software defects (i.e., bugs) are reported and required to be corrected or fixed. However, often some of reported defects will not be fixed when the software products are released, due to tight deadlines or limited resources [7]. After product release a small number of defects become "escalated" by customers, whose businesses are seriously impacted. Escalations of software defects require software vendors' immediate management attention and senior software engineers' immediate and continuous effort to reduce the business or financial loss to the customers. Therefore, software defect escalations are highly costly to the software vendors, with the associated costs amounting to millions of dollars each year. In addition, software defect escalations result in loss of reputation, satisfaction, loyalty and repeat revenue of customers, incurring extremely high costs in the long run for the enterprise software vendors [2, 3].

Due to time (i.e., deadlines) and resource limitations, enterprise software vendors can only fix a limited number of defects before product release. Thus, they must try to identify which reported defects have a high risk of escalation, which should be fixed at a much lower cost within the product development and testing cycle before product release. However, identifying software defects that are likely to escalate is a complex and difficult task. Software vendors often have in place human-centric processes for evaluating defect reports, but such processes are unreliable and subjective.

In this paper we propose a data-mining solution to predict escalation risks of defects to assist human experts in the review process of software defects. To the best of our knowledge, applying data mining for predicting software defect escalations is novel in software engineering. More specifically, we build an Escalation Prediction (EP) system that learns from history defects data and predicts escalation risk using data mining technology [1, 8, 12]. If the EP system can accurately predict the escalation risk of known defect reports, then many escalations will be prevented. This would save a huge amount of money for the enterprise software vendors [6].

Indeed, the ultimate business goal of EP (and many industrial applications using data mining) is to maximize the "net profit", that is, the difference in the cost before and after introducing the data mining solution, as opposed to the usual data-mining measures such as accuracy, AUC (area under the ROC curve), lift, or recall and precision [14]. However, the net profit is not equivalent to any of these standard machine learning measures

[5], and we have found little previous work that directly optimizes the net profit as the data mining effort.

We first set up a simple framework in which the problem of maximum net profit can be converted to minimum total cost in cost-sensitive learning under certain conditions (see Section 2). We then apply and compare several well-known cost-sensitive learning approaches on a defect report dataset to see how they perform, in terms of maximum net profit (Section 5). The EP system is shown to be able to improve greatly the net profit in software production. The system is currently deployed within the product groups of an enterprise software vendor, and it has quickly become a popular tool for prioritization defects for fixing. EP is one of the successful stories of data mining deployed in industry.

2. MAXIMUM NET PROFIT AND COST-SENSTITIVE LEARNING

In this section we will propose a novel method that converts the maximum net profit problem to cost-sensitive learning.

As discussed in the Section 1, correcting defects after an escalation occurs at the customer site is much more expensive than correcting them before product release. However, deadlines and limited resources usually allow only a small number of defects to be fixed at a lower cost before product release. If we treat escalated defects as positive examples and non-escalated defects as negative examples, then false negative, or *FN*, is when EP predicts negative (non-escalation) but the defect is actually positive (becomes escalated after product release). Thus, the cost of *FN* (correcting an escalated defect) is relatively high. On the other hand, false positive, or *FP*, is when EP predicts positive (escalation defects) but the defect is actually negative (non-escalation). Thus, the cost of *FP* (correcting a non-escalated defect) is relatively low (and is the same as the cost of true positive, *TP*, or fixing an escalated defect). The cost of true negative, *TN*, is zero.

We assume that the cost *FN* is 7 times as the cost of *FP*. Further, we assume that *FN* = \$7,000, and *FP* = \$1,000 (we did not use the actual numbers here for confidentiality reasons but the actual cost figures are some constant factor of the numbers used here). Then the cost metric can be represented in Table 1. As the cost metric is known, this would seem to be a straightforward cost-sensitive learning problem, in which the weighted misclassification cost is minimized.

Table 1: Cost metric.

	Actual Negative	Actual Positive
Predict Negative	0	7,000
Predict Positive	1,000	1,000

The problem is not that simple. The goal of the EP (Escalation Prediction) system (and many other real-world data mining applications) is to maximize the net profit after data mining is deployed. That is, we want to maximize the gain (or difference) with the data-mining effort compared to the previous, default practice. That is, we want to compare the cost (or profit) after data-mining based EP is deployed to some default policy before EP is deployed.

Let us first establish the formula for the net profit, which is the difference of the total costs before and after deploying EP. Let us first calculate the cost after deploying EP ("gross profit"). If the software vendor follows faithfully the EP's predictions, it will fix all defects predicted positively, and ignore all defects predicted negatively. The cost of correcting all defects predicted positively by EP is thus the multiplication of the number of defects predicated positively and the cost of correcting such a defect; that is, $(tp+fp)\times FP$. (Note that we use small letters, such as *tp*, to represent the number of true positive cases, and capital letters, such as *TP*, to represent the cost of true positive cases. Similar notations are used for other cases.)

After the software is released, the would-be escalated defects predicted negatively by EP (i.e., false negatives) will escalate and must be fixed at a much higher cost of $fn\times FN$. Thus, the total cost after deploying EP is the sum of the two costs described above, plus the cost of the data mining effort (such as the cost of the tool, computer and human cost of using the tool, etc). If we ignore the cost of the data mining effort now, then the total cost of deploying EP is:

$$(tp+fp)\times FP+fn\times FN. \qquad (1)$$

Assume that the default policy (before deploying EP) is to ignore all defect reports before software release, and then correct all escalated defects after release. Then using the same notation, the cost of this default policy is simply the cost of correcting all escalated defects. That is:

$$(tp+fn)\times FN. \qquad (2)$$

Thus, the net profit is to subtract (1) from (2). That is:

$$Net\ profit = tp\times(FN-FP)-fp\times FP = 6000\times tp-1000\times fp. \qquad (3)$$

On the other hand, if the default policy is to correct all of the defects (which rarely happens due to deadlines and scarce resource), the cost would be $(tp+fp+tn+fn)\times FP$. Subtracting (1) from the cost of the correcting-all policy above, the net profit would be: $1000\times tn-6000\times fn$. Thus, in general, the net profit varies with the default policy.

We propose a novel and simple approach to convert a formula of maximum net profit (such as (3) above) to cost-sensitive learning under certain conditions. Cost sensitive learning basically attempts to minimize the weighted cost, which can be expressed as:

$$tp\times TP + fp\times FP + tn\times TN + fn\times FN. \qquad (4)$$

Therefore, as long as the net profit, such as (3), can be expressed as a linear formula of *tp*, *fp*, *tn*, and *fn*, we can negate its coefficients in the linear formula, and re-assign the cost metric by the negated coefficients. For example, the cost metric for the net profit (3) can be converted and represented in Table 2. In the rest of the paper, we will study EP under this cost metric.

Comparing the two cost metrics in Tables 1 and 2, we can see that one can be converted into the other when the first row is subtracted to the second row [11]. However, even though the optimal prediction for minimal total cost remains the same after such conversions, the actual value of the total cost (4) would certainly be changed. That is, the software vendors not only want to know what predictions will produce the maximum net profit, but also the actual value of the net profit, as it reflects how much gain it may actually obtain after the data-mining based EP is deployed. Indeed, it is possible that the maximum net profit is negative (see Table 3 later in Section 5), indicating that it may be counter-productive to deploy the data mining system. In addition, as we have omitted the cost of the data mining effort (such as the tool cost, computer and human cost), the software vendor must obtain a positive net profit with a value that is large enough to offset other costs involved and decide if EP is a worthwhile endeavor or not.

To conclude, different default policies often result in different net profit calculations. However, under the condition that the net profit can be represented linearly by *tp*, *fp*, *tn*, and *fn*, we show that a simple and novel method can convert the maximum net profit to a new cost metric. Thus, the maximum net profit problem can now be solved by cost-sensitive learning. The result obtained (actual net profit value) will be useful for the software vendor to decide the overall benefit of deploying the data-mining system.

Table 2: Cost metric converted from (3). This cost metric will be used in the rest of the paper.

	Actual Negative	Actual Positive
Predict Negative	0	0
Predict Positive	1,000	–6,000

3. COST-SENSITIVE LEARNING

Cost-sensitive learning algorithms can be broadly categorized into two types. One is to design a *wrapper* that converts existing cost-insensitive (or cost-blind) base learning algorithms into cost-sensitive. The wrapper method is also called *cost-sensitive meta-learning*. The other is to design cost-sensitive learning algorithms directly [19, 10, 15]. In this section we will discuss briefly several well-known cost-sensitive learning approaches, including a method of our own (CSTree, described later) and several new improvements. Due to space limitation we will not provide full details of each method; instead, only highlights are given. The first four approaches belong to the wrapper method; they apply sampling etc. on cost-blind base algorithms to make them cost-sensitive. The last method is cost-sensitive by nature. These approaches will be compared experimentally in the next section to see which one(s) can reliably obtain the maximum net profit in Escalation Prediction.

The first popular approach, called "Undersampling" in this paper, is to rebalance the training data by sampling the data sets. Previous work (see, e.g., [10]) has indicated that over-sampling the minority class can cause overfitting, and may not be helpful to improve predictive performance. Other research [20] shows that keeping all examples of the rare class, and under-sampling the majority class to be about the same as the rare class performs well measured by AUC. Thus, we will use this simple strategy. That is, we keep all positive examples (escalated defects), and randomly sample without replacement the negative examples such that the two classes are balanced.

However, as the positive examples occupy only about 1% of the original training and test sets (see Section 4), the balanced dataset contains only about 2% of the original training set, which is very small. Therefore, we apply bagging [4] to Undersampling, as bagging has been shown to be effective in improving predictive accuracy and probability estimation measured by AUC. Bagging is also applied to other methods (see later).

As Undersampling itself is not cost-sensitive, the class label produced cannot be used directly in predicting escalation for maximum net profit. As the base learning algorithms used also produce probability estimates for the labels, testing cases can be ranked. To accurately calculate the maximum net profit, the threshold method is used: a separate validation set is used to find the best threshold on the probability estimates to classify the validation cases that would produce the maximum net profit (on the validation set), and this best threshold is then applied to the test set to classify the test cases into positive (if the predicted probability is larger than or equal to the threshold) or negative (if the predicted probability is less than the threshold). The net profit on the test set can then be calculated and obtained.

The second approach, called "Costing", is an advanced sampling which weighs examples according to the cost, producing cost-sensitivity with a provable performance guarantee [22]. The advanced sampling used is rejection sampling, in which each example in the original training set is drawn once, and is accepted into the sample with the probability proportional to its "importance". The size of the re-sampled dataset is smaller than the original set. With bagging, the method is shown to outperform other methods measured by the maximum total profit in two real-world datasets [22].

The third approach, called "Relabeling", relabels the classes of instances by applying the minimum expected cost criterion [9]. MetaCost [9] belongs to this approach. MetaCost uses bagging as the ensemble method.

The forth approach, called "Weighting" [18], induces cost-sensitivity by integrating the instances' weights directly into the classifier building process. It works when the base learners can accept weights directly. This method does not rely on bagging. However, to compare with other cost-sensitive methods, we also apply bagging to this method.

The fifth approach, called "CSTree", is a recently proposed cost-sensitive decision-tree algorithm that uses the total cost directly as a split criterion [15]. That is, instead of minimizing entropy in the tree building process, the split criterion simply minimizes the total misclassification cost given a cost metric. CSTree is, by nature, truly cost-sensitive, and thus, the result of classifying the test cases is used directly in the calculation of the maximum net profit.

Several improvements have been made in the CSTree for the EP application. First of all, compared to [15], the costs for true positive (*TP*) and true negative (*TN*) may not be zero here. Second, *expected* total cost reduction is used during tree building. That is, an attribute may be selected as a root if the expected cost reduction is maximum (among other attributes), and is greater than 0. More specifically, for a given set of cases without split, if C_P (= $tp \times TP + fp \times FP$) is the total cost of labeling a leaf positively, and C_N (= $tn \times TN + fn \times FN$) is the total cost of labeling a leaf negatively, then the probability of a positive outcome is first estimated by the relative cost of C_P and C_N as

$1-\frac{C_P}{C_P+C_N}=\frac{C_N}{C_P+C_N}$. (The smaller the cost C_P, the larger the probability of being positive, as minimum cost is sought). The expected misclassification cost of being positive is thus: $E_P=\frac{C_N}{C_P+C_N}\times C_P$. Similarly, the probability of a negative outcome is $\frac{C_P}{C_P+C_N}$, and the expected misclassification cost of being negative is: $E_N=\frac{C_P}{C_P+C_N}\times C_N$. Therefore, without splitting, the expected total misclassification cost of a given set of examples is: $E=E_P+E_N=\frac{2\times C_P\times C_N}{C_P+C_N}$. If an attribute A has k branches, then the expected total misclassification cost after splitting on A is: $E_A=2\times\sum_{i=1}^{k}\frac{C_{P_i}\times C_{N_i}}{C_{P_i}+C_{N_i}}$. Thus, ($E$ – E_A) is the expected cost reduction splitting on A. During tree construction, an attribute with the maximum expected cost reduction is chosen, and if the value is greater than 0, then the attribute is chosen to split the training examples; if not, a leaf node is formed.

The last improvement is tree pruning. The original method did not incorporate pruning. We have found that without pruning, the tree overfits the training examples. This means that a larger tree (that fits the training examples better) predicts worse compared to a small tree. We have implemented post pruning similar to the post-pruning in C4.5 [17] in our cost-sensitive decision tree (but our post-pruning is guided by minimum total misclassification cost).

The CSTree has two distinctive advantages. The first one is that there is no need for sampling in the training data. The original training data can be used directly in building the decision tree. That is, CSTree can naturally utilize all information in the training data. Second, CSTree does not rely on bagging for the final outcome. This implies that the final results can be comprehensible by domain experts.

4. THE DATASETS

Our dataset consists of historical defect reports from industry software projects of an enterprise software vendor. Defect reports change over time and so there is an opportunity to learn from multiple different versions of a single defect report. Additionally, the same defect can be reported several times by different parties. Therefore, numerous data records in the dataset may belong to only one defect. Confidentiality of the data only allows us to give a brief description of the data. The data was collected in a period of 2004, and contains a total of about 165,000 records (defect report observations). Note that the class labels of this dataset have all been verified by real escalations of the software products, thus this dataset contains much less noise than the one used in our previous work [16]. The total number of attributes is 53, most of which are numerical attributes (including dates), and a few are nominal (such as product name). The system uses three types of inputs: 1) raw inputs which correspond to field values which are reported by users and stored in the defect tracking database, 2) row-level transformations of raw values, such as concatenating two string-valued raw inputs into a derived new string-valued input, and 3) statistical inputs which are derived from statistical analyses of all rows which fall into a given time period, such as a fiscal quarter.

Statistical inputs were particularly useful for string valued raw inputs because string-valued data is typically difficult to use for machine learning. For example, a raw input for "CUSTOMER_NAME" may have 20,000 possible customer name values. Therefore, the system calculates statistics for string-valued inputs. These statistics include counts (number of defects submitted by customer "abc", and number of escalations raised by customer "abc"), means (number of escalations over number of defects submitted for hardware platform "xyz"), and the probability of an escalation, and so on. Further analysis of these inputs and their utility for prediction is of great interest but beyond the scope of this paper.

The target attribute is binary (escalation or no escalation). The whole dataset is split up into training and test sets according to the defect report date. The training set contains about two thirds of the records (before a certain date), and the test set contains one third of the records (after that date). The dataset is very imbalanced, with slightly less than 1% of the positive examples (escalated defects).

5. COMPARING COST-SENSITIVE LEARNING APPROACHES FOR EP

In this section we compare the five cost-sensitive learning approaches (Undersampling, Costing, Relabeling, Weighting, and CSTree) discussed in Section 3 on the latest dataset we have obtained. Again the latest dataset contains much less noise than the dataset used in our previous work [16], thus the results reported here are different (but the general conclusion is still the same). In addition, more base learning algorithms are included in the current study (see later).

Again the first four approaches are wrappers or meta cost-sensitive learning methods, as they apply to any cost-blind base learning algorithms to make them cost-sensitive. We choose four popular learning algorithms, naïve Bayes, the decision tree algorithm C4.5 [17], decision stump [13], and REPTree [21] as their base learners, due to their popularity and high efficiency. The fifth method (CSTree) is also a decision tree but it is cost-sensitive in nature. It would be interesting to compare various wrapper methods applying to cost-blind C4.5 to the cost-sensitive decision tree (CSTree).

To make the net profit result comparable with different approaches, we use “unit” net profit, defined as the net profit divided by the number of records in the test sets in our comparison. Note that even if a data mining method obtains a unit net profit of $1, the saving to the enterprise software vendors can be quite significant. If we assume that 1 unit represents $50 in reality (a reasonable figure), then with 1,000 software defect reports in a software product to be released, a total saving with EP’s prediction would be 1,000×50 = $50,000 for a typical customer. Alternatively, assuming a unit cost of $250 and 50,000 defect reports the total savings amount to $12.5 million.

Bagging is applied to all methods. The maximum unit net profit, when the bagging iteration is set to 1 (no bagging), 10, and 100 are shown in Table 3.

Table 3: Comparing unit net profit using five different cost-sensitive approaches.

Cost-sensitive Approaches		**Bagging Iterations**		
		1	**10**	**100**
Undersampling	Naïve Bayes	0.00	0.43	1.85
	C4.5	-0.05	0.01	3.52
	REPTree	0.00	6.97	10.80
	DecisionStump	0.00	0.00	0.00
Costing	Naïve Bayes	-6.60	-5.45	-5.26
	C4.5	11.87	13.85	13.99
	REPTree	9.63	11.31	13.34
	DecisionStump	0.00	0.00	0.00
Relabeling (MetaCost)	Naïve Bayes	-16.49	-17.56	-18.01
	C4.5	0.00	0.00	0.00
	REPTree	8.00	12.83	11.15
	DecisionStump	0.00	0.00	0.00
Weighting	Naïve Bayes	-5.69	-6.10	-6.93
	C4.5	9.10	11.38	11.47
	REPTree	12.96	14.16	14.39
	DecisionStump	0.00	0.00	0.00
CSTree	NA	14.72	14.15	14.07

From the results in the table, we can draw several interesting conclusions.

First, in terms of performance of the five different cost-sensitive learning algorithms, we can see that they perform quite differently, even when they use the same base learning algorithms. This means that some trial-and-error is needed to choose the best performing cost-sensitive approach in EP (and other real-world applications). For meta-learning approaches (the first four approaches), different base learners also yield very different performance. In general, C4.5 and REPTree perform better than other base learning algorithms. The fifth method, CSTree, is the best. Decision tree models (such as C4.5 and REPTree) with no bagging (bagging iteration is 1) also produce comprehensible results that can provide insights to the software vendors as to why certain defects are highly likely to escalate.

Second, compared to no bagging (bagging iteration is 1), bagging (with 10 or 100 iterations) generally improves the result when the maximum unit net profit is positive. However, CSTree achieves an excellent result without bagging (with the highest unit net profit of 14.72). Again without the need of bagging, a single tree has the advantage of producing comprehensible results for domain experts to exam the outcome of data mining.

Third, sometimes cost-sensitive methods produce very negative in terms of unit net profit, such as Costing, Relabeling, and Weighting with naïve Bayes. As we discussed in Section 2, often these absolute values of the unit net profit is important for software enterprise vendors to decide if the deployment of data mining is profitable or not, and by how much, compared to the default strategy. Negative net profit, even though it is a maximal value, indicates that it is not worthwhile to deploy the data mining effort. Fortunately in our case, we have found clear winners: the CSTree, Costing, and Weighting, as they have resulted in large positive unit net profit. Deployment of EP with CSTree is encouraging. See Section 6 for details.

To summarize, of all algorithms tested, CSTree performs best for the EP task. In Table 4, we rank the five algorithms according to their highest unit net profit, and the average unit net profit (averaging over the four base learners for the meta cost-sensitive learning methods). Both rankings indicate that CSTree performs the best, followed by Costing or Weighting, and followed by Undersampling or Relabeling.

Table 4: The rankings of the five cost-sensitive methods in comparison. Numbers in parentheses are the unit net profit.

	1 (best)	2	3	4	5
Ranked by highest profit	CSTree (14.72)	Weighting (14.39)	Costing (13.99)	Relabeling (12.83)	Undersampling (10.80)
Ranked by average profit	CSTree (14.31)	Costing (4.72)	Weighting (4.56)	Undersampling (1.96)	Relabeling (-1.67)

Clearly CSTree without bagging is the most appropriate choice for deployment in the software development cycle, due to its superior performance and comprehensibility. We will discuss the deployment of EP in the next section.

6. DEPLOYMENT

Our EP system has been in deployment for many months in the product group of an enterprise software vendor where the dataset comes from. It has been used to make suggestions on current defect reports with high risks of escalation. As the software vendor must wait to see which defect reports are actually escalated after the product is released, and the would-be escalations will not happen after they are accurately predicted by EP and corrected before software release, the final evaluation cannot be obtained until the software has been released and has been in use by customers for some period of time (in the order of one or more quarters).

We have evaluated EP using the defect reports submitted or updated during the most recent three weeks in the test set. Any records corresponding to defect reports which had already been escalated at the time of preparing the data set were also removed. After EP makes its predictions, the results are compared to the actual escalations happened up to date. The EP prediction performance is quite good. The lift chart of EP's prediction is well above the random diagonal line: at top 10 percentile (among top 10% of the most likely predicted escalations), about 70% of the actual escalations are predicted; at 20 percentile about 85%; and at 30 percentile about 90%. These results are of significant business value to the software vendor. In addition, the product group has provided positive feedback on the performance of the EP system, citing that it "catches" defect reports that, after thorough evaluation by specialist, are considered likely candidates for future escalations. For instance, some defect reports have been found to have been assigned a lower than appropriate priority. After a prediction of high escalation risk becomes available such a defect report can be upgraded to a higher priority which will lead to expedited evaluation and resolution.

7. CONCLUSIONS

Enterprise software development is an extremely complex process with hundreds or even thousands of defects that need to be prioritized and resolved, preventing them from escalation by customers and incurring a very high cost to the software vendors. In this paper, we present a successful case for predicting escalation risks mined from known product defect reports. The enterprise software vendors can proactively resolve these defects with the greatest risk of escalation at a much lower cost. This can save software vendors an enormous amount of software maintenance cost. An escalation prediction (EP) system based on data-mining for the maximum net profit has been proposed and tested, and is currently deployed at an enterprise software vendor. Results provide strong evidence that we can indeed make useful predictions about the escalation risk of product defects.

More specifically, we establish a general framework in which the maximum net profit problem can be converted to cost-sensitive learning. We then evaluate and compare five major cost-sensitive learning methods for their effectiveness for EP. We find that the CSTree (with novel improvements) can produce large positive unit net profit, as well as comprehensible results. This is often important for deploying data mining solutions in industry.

In our future work, we plan to continue to improve the effectiveness of the EP system and track its results from the software vendors.

8. REFERENCES

[1] Berry, M.J.A., and Linoff, G. 1997. *Data Mining Techniques: For Marketing, Sales, and Customer Support*. John Wiley & Sons.

[2] Boehm, B.W., and Basili, V. 2001. *Software Defect Reduction Top 10 List*. Computer 34(1): 135-137.

[3] Boehm, B.W. 1981. *Software Engineering Economics*. Prentice-Hall Advances in Computing Science & Technology Series.

[4] Breiman, L. 1996. Bagging Predictors. *Machine Learning* 24(2): 123-140.

[5] Bruckhaus, T. 2006 (forthcoming). The Business Impact of Predictive Analytics. Book chapter in *Knowledge Discovery and Data Mining: Challenges and Realities with Real World Data*. Zhu, Q, and Davidson, I., editors. Idea Group Publishing, Hershey, PA

[6] Bruckhaus, T., Ling, C.X., Madhavji, N.H., and Sheng, S. 2004. Software Escalation Prediction with Data Mining. *Workshop on Predictive Software Models*, A STEP Software Technology & Engineering Practice.

[7] Chulani, S., and Boehm, B.W. 1997. *Modeling Software Defect Introduction*. California Software Symposium, Nov.

[8] Dai, H. (editor). 2003. Proceedings of the International Workshop on Data Mining for Software Engineering and Knowledge Engineering.

[9] Domingos, P. 1999. MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 155-164, ACM Press.

[10] Drummond, C., and Holte, R.C. 2003. C4.5, Class Imbalance, and Cost Sensitivity: Why under-sampling beats over-sampling. *Workshop on Learning from Imbalanced Datasets II.*

[11] Elkan, C. 2001. The Foundations of Cost-Sensitive Learning. In *Proceedings of the International Joint Conference of Artificial Intelligence*, 973-978.

[12] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (editors). 1996. *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press.

[13] Iba, W. and Langley, P. 1992. Induction of One-level Decision Trees. In *Proceedings of the Ninth International Workshop on Machine Learning*, 233-240.

[14] Ling, C.X., and Li, C. 1998. Data Mining for Direct Marketing: Specific Problems and Solutions. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, 73-79.

[15] Ling, C.X., Yang, Q., Wang, J., and Zhang, S. 2004. Decision trees with minimal costs. In *Proceedings of International Conference on Machine Learning*.

[16] Ling, C.X., Sheng, S. Bruckhaus, T., and Madhavji, N.H. 2005. Predicting Software Escalations with Maximum ROI. In *Proceedings of International Conference of Data Mining*.

[17] Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

[18] Ting, K.M. 2002. An Instance-Weighting Method to Induce Cost-Sensitive Trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):659-665.

[19] Turney, P.D. 1995. Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm. *Journal of Artificial Intelligence Research* 2:369-409.

[20] Weiss, G., and Provost, F. 2003. Learning when Training Data are Costly: The Effect of Class Distribution on Tree Induction. *Journal of Artificial Intelligence Research* 19: 315-354.

[21] Witten, I.H., and Frank, E. 2000. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco.

[22] Zadrozny, B., Langford, J., and Abe, N. 2003. Cost-Sensitive Learning by Cost-Proportionate Example Weighting. In *Proceedings of International Conference of Data Mining*.