Data Mining
COT 6930
Meta Learning Schemes:
Class Handout

Erik Geleyn*

Florida Atlantic University
Boca Raton, Florida USA

February 2002

Abstract

This handout is to be used in complement to the two lectures I will be giving on February, 26^{th} and 28^{th}. This handout contains the outline of the lectures as well as some material to improve the understanding and practical experience of the students. Please print a copy of the handout and bring it to the lectures.

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory, Dept. of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800, Email: taghi@cse.fau.edu, URL: www.cse.fau.edu/esel/. For classroom use only, for other use please contact the authors. .

1 Introduction to meta learning schemes

This handout presents recent techniques to improve the performances of existing software classification models. We will present a set of techniques from the data mining and machine learning field known as meta learning schemes. The idea is to combine several classification models to improve the classification accuracy by compensating each model's weaknesses. In the same way wise people make critical decisions by consulting with a panel of experts rather than relying on their sole judgement or on a single expert [20]; We will use combined classification models (experts) to improve the combined decision of individual models.

1.1 Concepts

Several techniques have been proposed in recent years to implement this concept of expert combined decisions. First of all, there is a prerequisite for any classification method to be eligible for a combined decision process: it must be unstable [2, 4, 11]. In the same way there is little interest in attending a plebiscite in a dictatorship where each expert (voter) is going to provide the same desired outcome (a strong support to the dictatorial leader); there is little interest in combining decisions of very similar classifiers. A classifier is defined as unstable if slight changes in the training data produce significantly different classifiers. Examples of unstable classifiers typically include decision trees or neural nets [4], while stable classification methods include Case-Based Reasoning (CBR) [2] or discriminant analysis.

Other concepts that need to be understood before investigating this field are the definitions of a *weak* and a *strong* learner. A weak learner is defined as a learner that is only a little better than guessing or random (the misclassification rate is close to 50%) [9]. A strong learner is defined as a learner that produces a classifier with a reasonable classification accuracy. Examples of weak learners include one level decision trees, while C4.5 may be considered as a strong learner.

1.2 Advantages

The different meta learning schemes have been applied successfully on some UCI Machine Learning Repository data sets [6, 13, 14]. They were also used in various prediction activities such as transaction fraud detection systems [11], or text mining for an online banking application [19]. Meta learning schemes have also been used in outlier detection and in identifying mislabelled training data [5].

Researchers have been able to highlight many advantages of combined classifiers over simple classifiers. The main advantage is the performance increase noted on data sets from the UCI Machine Learning Repository data sets and on real world applications [7, 10, 19, 11, 13, 14]. Some researchers were even able to present some guaranties about the level of accuracy of their algorithms [9].

Another advantage of these combined "decision makers" is that they are less prone to overfitting [9]. The reason is quite obvious: each classifier is trained on separate data and the combined learner is not likely to overfit the original training data [20]. Overfitting describes a situation where a learner performs well on its training data but not on fresh data. Unpruned decision trees are likely to overfit their training data.

The absence of tuning to achieve good classification results [9] is also an obvious advantage of these methods. We can hardly improve on Breiman in [4]: "Read in the data, press the start button." The obvious advantage here is the ease of model selection. We no longer need to spend hours finding the best values for parameters, such as the number of nearest neighbors in Case-Based Reasoning or the pruning factors in decision trees. These tasks are time consuming and require expertise in the algorithm you are using.

1.3 Drawbacks

These methods also hold a few drawbacks. The main criticism of these combined method is that it is very hard to interpret the decisions made by the combined model. While interpreting a single decision tree is fairly simple, interpreting a combination of 10 different trees is almost impossible. The practitioner chooses to sacrifice the interpretation of the prediction to privilege performance and simplicity.

Another drawback is the computational cost of these methods. If you are combining 10 decision trees and using 10-fold cross validation to evaluate your method, you need to build 100 trees. You may have noticed this effect when running your experiments for Part IV of the homework.

The rest of the handout will describe and show you some practical examples for Bagging, Boosting and Cost-Boosting.

2 Bagging

The simplest way to combine learners in a meta learner is to randomly resample from the original training dataset, build a learner for each resampled dataset, use the prediction of each learner in a simple vote (or an unweighed average if we are doing prediction) to obtain the combined decision on fresh data. This technique is known as *Bagging*. Bagging stands for bootstrap aggregating [2].

2.1 Concepts

In Bagging, the resampling is performed with replacement: an instance may appear more than once, others may not be resampled. The resampled data sets have usually the same size as the original training data set. The combined decision or *final hypothesis* for classification is obtained using an unweighed vote.

The algorithm used for the combined decision is referred to as the weak learner. One requirement for the weak learner is that it has to be unstable [2, 20]. The unstability of the weak learner ensures that small changes in the training data will yield significantly different learners. There is no point in combining very similar learners since they will provide very similar outcomes. Decision trees and neural networks are typical unstable learners [3].

The main advantage of the Bagging algorithm is its simplicity of implementation. The Bagging process is also a dream for parallel architectures since each model can be build independently from each other. Bagging is usually able to improve weak learners in a significant way. However, Bagging relies on random samples to generate the different learners and does not take into account the history of misclassifications or the quality of the trees.

Table 1: Notation of Symbols used in Bagging

Symbol	*Description*
h_t	Weak hypothesis on the t^{th} iteration
$h_t(x_i)$	Value of the weak hypothesis on instance i
$h_{fin}(x_i)$	Final hypothesis
m	Number of instances in training dataset
S	A Training dataset
T	Number of iterations
X	An instance space
x_i	An instance in an instance space X
Y	Class space
y_i	A class in Y

2.2 Algorithm

The complete Bagging algorithm as well as the necessary notations are given in Figure 1 and Table 1, respectively.

2.3 Let's do it by hand

In this part we will go through a simple example of the voting process in the Bagging algorithm.

An example of a combined vote is given in Table 2, where x_i and y_i represent the instance i and its actual class (-1 or 1) respectively. $h_j(i)$ represents the predicted class for instance x_i of a weak learner determined on the j^{th} iterations of a Bagging algorithm. Complete $h_{fin}(x_i)$ representing the combined decision of the 5 weak learners for instance x_i. Use majority voting to achieve the combined decision.

1. **Input**:
   - A data set S of order pairs $(x_1, y_1), ..., (x_m, y_m)$, where $x_i \in X$ is an instance space, $y_i \in Y = \{-1, +1\}$
   - Weak learning algorithm
   - An integer T specifying the number of iterations
2. **Do for** $t = 1, 2, ..., T$
   - Form a data set S_t by sampling n instances with replacement from the training data set S
   - Call Weak Learner, providing it with the distribution S_t
   - Get back a hypothesis $h_t : X \rightarrow Y$.
3. **Output** the final Hypothesis: $h_{fin}(x) = sign(\sum_{t=1}^{T} h_t(x))$

Figure 1: Bagging Algorithm

Table 2: Example of Voting Process

x_i	y_i	$h_1(i)$	$h_2(i)$	$h_3(i)$	$h_4(i)$	$h_5(i)$	$h_{fin}(x_i)$
x_1	1	1	1	1	-1	1	
x_2	1	1	-1	1	1	1	
x_3	-1	-1	1	1	1	-1	
x_4	1	-1	1	1	1	1	
x_5	1	1	1	1	-1	1	
x_6	-1	1	-1	-1	-1	-1	
x_7	-1	-1	-1	-1	1	-1	
x_8	-1	-1	-1	1	-1	-1	
x_9	-1	-1	-1	-1	-1	-1	
x_{10}	1	1	1	1	1	1	

3 Boosting

One shortcoming of Bagging is that it does not take into account the performances of the previous learners in its iteration process. This issue has been addressed by Freund and Schapire in [9, 14, 10]. They presented another meta learning scheme referred to as *Boosting*. Compared to Bagging, Boosting takes into account the results of previous classifiers by reweighing the instances in the original training data set according to the history of misclassifications.

3.1 Concepts

An instance that has been previously misclassified will have its weight increased and the next classifier will be encouraged to pay more attention to this instance than to a previously correctly classified instance, whose weight will have been decreased. Here, we assume that the learner is able to handle weighted instances.

In Boosting the weight updating is straightforward: the weight of correctly classified instances is decreased while the weight of misclassified instances is increased. The next learner will then focus on the *hard* to classify correctly instances with high weights. Thus each instance's weight holds the history of the previous classification (correct, incorrect).

In experiments, Boosting algorithms have already shown increased performances over Bagging algorithms [9, 13, 8, 18, 10, 15, 16, 12, 7, 17]. When boosting improves a classifier, it does it in a significant way. However some experiments have shown that in some cases the boosting algorithm could penalize a strong learner, worsening the classification accuracy. Experiments have shown that Boosting is not always able to improve classifiers, especially strong learners, but when it does it is with substantial

Table 3: Notation of Symbols used in AdaBoost

Symbol	*Description*
α_t	Parameter choosen as a weight for weak hypothesis h_t
$D_t(i)$	Distribution used as a weight for instance i on iteration t
$D_{t+1}(i)$	Distribution used as a weight for instance i on iteration $t+1$
ϵ_t	Error of the weak hypothesis h_t
h_t	Weak hypothesis on the t^{th} iteration
$h_t(x_i)$	Value of the weak hypothesis on instance x_i
$h_{fin}(x_i)$	Final hypothesis
m	Number of instances in training data set
T	Number of iterations
X	An instance space
x_i	An instance in an instance space X
Y	Class space
y_i	A class in Y
Z_t	Normalization constant to ensure that D_{t+1} will be a distribution

benefit [10].

3.2 Algorithm

The complete Boosting algorithm (AdaBoost) presented by Freund and Schapire [9] as well as the necessary notations are given in Figure 2 and Table 3, respectively.

3.3 Handling a weighted Dataset

We assumed earlier that the learners were able to handle weighted datasets. This is not always true. One way around this is to resample from the original training data set with repetition, giving a higher probability of selection to the instances with higher weights; The resampled training data set is then provided to the learner [20].

The resampled training data set is obtained from the original data set by resampling

1. **Input**:
    - A set of order pairs $(x_1, y_1), ..., (x_m, y_m)$, where $x_i \in X$ is an instance space, $y_i \in Y = \{-1, +1\}$
    - Weak learning algorithm
    - An integer T specifying the number of iterations
2. **Initialize** $D_1(i) = 1/m$ for all i.
3. **Do for** $t = 1, 2, ..., T$
    - Call Weak Learner, providing it with the distribution D_t
    - Get back a hypothesis $h_t : X \rightarrow Y$.
    - Calculate the error of $h_t : \epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$. If $\epsilon_t > \frac{1}{2}$, then set $T = t - 1$ and abort loop.
    - Set $\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$
    - Update distribution $D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$
      where Z_t is a normalization constant(chosen so that D_{t+1} will be a distribution).
4. **Output** the final Hypothesis: $h_{fin}(x_i) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x_i)\right)$

Figure 2: AdaBoost Algorithm

with replacement. The instances are sampled according to their weights: the higher the weight, the higher the probability of being selected. Typically, instances with higher weights will be selected several times and the resampled training data will hold several copies of this instance. In the same way instances with low weights may not appear at all in the resampled training data. The resampled training data is then passed to the learner. The learner will focus on the instances that have several duplicates thus ensuring a better classification for instances with higher weights in the training data.

The algorithm we present for probability sampling is “Stochastic Sampling with Replacement.” [1] The idea of this method may be explained as follows: imagine a spinning roulette where each slot is given a size according to its weight (instance with large weights are given larger slots). Hence when spinning the roulette the probability of an instance of being selected derives directly from its weight.

Table 4 shows a list of five instances and their associated weight. Determine the angle of the slot associated to each instance.

Complete Figure 3 with each slot associated to each instance. Write a program to generate 10 random values between 1 and 360. The output may be the following: 165, 327, 48, 348, 128, 142, 230, 337, 11, and 106.

Use a protractor and determine the resampled dataset.

Table 4: Example of Stochastic Sampling with replacement: Instance weights

Instance	Weight	Slot Angle (degrees)
1	0.0555	
2	0.0278	
3	0.1111	
4	0.1111	
5	0.4444	
6	0.0278	
7	0.1111	
8	0.0555	
9	0.0278	
10	0.0278	

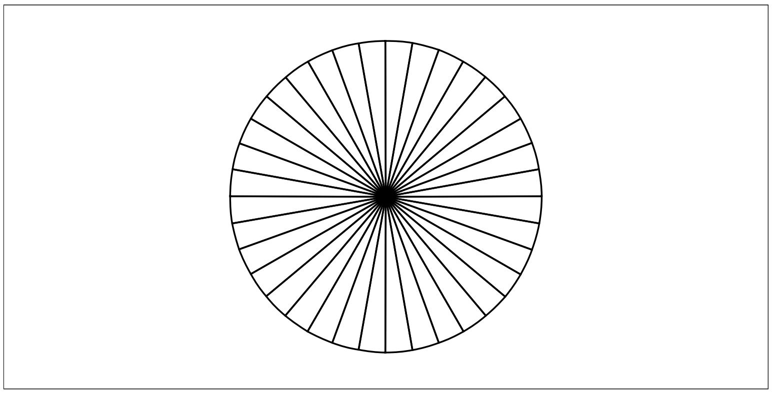

Figure 3: Example of Stochastic Sampling with replacement: Generated Roulette

3.4 Let's do it by hand

We will know perform an example of weight update using the boosting algorithm. Table 5 presents a weight update using the Boosting algorithm. $D_j(i)$ represents the weight for instance x_i for the j^{th} iteration of the Boosting algorithm. The actual class (-1, 1) of the instance x_i is given by y_i. Please note that the initial weights are set to $1/N$ (0.1), where N represents the initial number of instances in the training data. $h_1(i)$ represents the predicted output of the first learner trained with the original fit data set.

Update the weights according to the correct ($y_i = h_1(i)$) or incorrect ($y_i \neq h_1(i)$) classification of the instances and the Adaboost algorithm described in Table 3.2.

1. First, compute the value of ϵ_t.

$$\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i) = \tag{1}$$

2. Then determine α_t

$$\alpha_t = \frac{1}{2} ln \frac{1 - \epsilon_t}{\epsilon_t} = \tag{2}$$

3. Complete the $e^{\pm\alpha_1}$ column according to the correct or incorrect classification.

Table 5: Example of AdaBoost Update Process

$D_1(i)$	x_i	y_i	$h_1(i)$	$e^{\pm\alpha_1}$	$D_1(i) \times e^{\pm\alpha_1}$	$D_2(i)$
0.1000	x_1	1	1			
0.1000	x_2	-1	1			
0.1000	x_3	1	-1			
0.1000	x_4	1	-1			
0.1000	x_5	1	1			
0.1000	x_6	-1	1			
0.1000	x_7	-1	-1			
0.1000	x_8	-1	-1			
0.1000	x_9	-1	-1			
0.1000	x_{10}	1	1			
					Z_t=	

4. Compute $D_1(i) \times e^{\pm\alpha_1}$ and complete the table.

5. Finally, determine the new weights by computing Z_t and normalizing.

You should notice that all the instances correctly classified have seen their weights decreased and the misclassified ones have had their weights increased. The new weights are then provided to the next learner along with the training data set so it can focus on the previously misclassified instances.

4 Cost-Boosting

One shortcoming of the boosting algorithm presented earlier is that the algorithm does not take into account the eventual cost of misclassification in their weight updating rules. Thus, the weight increases or decreases is the same for every type of misclassifications.

Let us recall the specificity of the software quality classification problem. A Type I error (or misclassification) occurs when a not fault-prone module is classified as fault-prone and a Type II error occurs when a fault-prone module is classified as not fault-prone. Obviously the cost associated with a Type II misclassification is much higher than the one associated with a Type I misclassification. A Type II misclassification may allow faults to remain in the product. Given these specificity, the cost associated with each misclassification is of primary concerns to the practitioners in our field.

4.1 Concepts

The only way to induce cost sensitivity with the original boosting algorithm is through the weak learner by using a cost matrix. One other way to handle the problem is to modify the boosting algorithm to allow the weight updates to reflect the cost of each type of misclassifications. Typically we would want the learners to focus on the costly and hard to classify instances and not only on the hard to classify instances.

Ting and Zheng [16] proposed CostBoosting to tackle the cost-sensitivity issue. CostBoosting is similar to AdaBoost, only the cost of misclassification are taken into account during the weight update phase and in the computation of the final hypothesis.

4.2 Algorithm

Figure 4 describes the CostBoosting algorithm along with the notation provided in Table 6.

4.3 Let's do it by hand

1. **Input**:

   - A set of order pairs $(x_1, y_1), ..., (x_m, y_m)$, where $x_i \in X$ is an instance space, $y_i \in Y = \{-1, +1\}$
   - Weak learning algorithm
   - An integer T specifying the number of iterations

2. **Initialize** $D_1(i) = 1/m$ for all i.

3. **Do for** $t = 1, 2, ..., T$

   - Call Weak Learner, providing it with the distribution D_t
   - Get back a hypothesis $h_t : X \rightarrow Y$.
   - Calculate the error of $h_t : \epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$. If $\epsilon_t > \frac{1}{2}$, then set $T = t - 1$ and abort loop.
   - Set $\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$
   - Update distribution $D_t : D_{t+1}(i) = \frac{D'_{t+1}(i)}{\sum_i^m D'_{t+1}(i)}$

   $$D'_{t+1}(i) = \begin{cases} cost(actual(i), predicted(i)) & \text{if } actual(i) \neq predicted(i) \\ mD_t(i) & \text{otherwise.} \end{cases}$$

4. **Output** the final Hypothesis: $h_{fin}(x_i) = min \sum_k^K \sum_{t=1}^T |\alpha_t h_t(x_i) cost(k, j)|$, where K is the total number of classes; and $cost(k, j)$ is the misclassification cost of classifying a class k instance as class j.

Figure 4: Cost-Boosting Algorithm

Table 6: Notation of Symbols used in Cost-Boosting

Symbol	*Description*
α_t	Parameter chosen as a weight for weak hypothesis h_t
$cost(k,j)$	Misclassification cost of classifying a class k instance as class j
$D_t(i)$	Distribution used as a weight for instance i on iteration t
$D_{t+1}(i)$	Distribution used as a weight for instance $i+1$ on iteration $t+1$
$D'_{t+1}(i)$	Cost adjustment factor used to determine D_{t+1}.
h_t	Weak hypothesis on the t^{th} iteration
$h_t(x_i)$	Value of the weak hypothesis on instance x_i
$h_{fin}(x_i)$	Final hypothesis
K	Total number of classes
m	Number of instances in training data set
T	Number of iterations
X	An instance space
x_i	An instance in an instance space X
Y	Class space
y_i	A class in Y

We will know perform an example of weight update using the CostBoosting algorithm. Table 7 presents a weight update using the Boosting algorithm. $D_j(i)$ represents the weight for instance x_i for the j^{th} iteration of the Boosting algorithm. The actual class (-1, 1) of the instance x_i is given by y_i. Please note that the initial weights are set to $1/N$ (0.1), where N represents the initial number of instances in the training data. $h_1(i)$ represents the predicted output of the first learner trained with the original fit data set. We attached a cost of 1 to misclassified 1 classes and a weight of 2 to the misclassified -1 classes.

Update the weights for two consecutive runs according to the correct $(y_i = h_1(i))$ or incorrect $(y_i \neq h_1(i))$ classification of the instances and the Adaboost algorithm described in Table 4.2.

1. First, compute the value of ϵ_t.

$$\epsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t = \tag{3}$$

2. Then determine α_t

$$\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t} = \tag{4}$$

3. Compute $D'_{t+1}(i)$ according to the classification:

4. Finally, determine the new weights $D_{t+1}(i)$

You should notice that misclassified -1 instances have their weights increased more aggressively than misclassified 1 instances.

Table 7: Example of CostBoosting Update Process

i	y_i	$D_1(i)$	$h_1(i)$	$D'_2(i)$	$D_2(i)$	$h_2(i)$	$D'_3(i)$	$D_3(i)$
1	1	0.067	1			1		
2	1	0.067	-1			1		
3	1	0.067	1			1		
4	1	0.067	-1			-1		
5	1	0.067	-1			1		
6	1	0.067	1			-1		
7	1	0.067	1			1		
8	1	0.067	-1			-1		
9	1	0.067	1			1		
10	1	0.067	1			1		
11	-1	0.067	-1			-1		
12	-1	0.067	-1			-1		
13	-1	0.067	-1			1		
14	-1	0.067	1			-1		
15	-1	0.067	-1			-1		

5 Summary

Combining models to derive a prediction provides many advantages. They usually increase the performance when compared to a single learner. They allow weak learners to achieve good performances. They also reduce the risk of overfitting.

We presented three methods in this handout: Bagging, Boosting and CostBoosting. Bagging used randomly resampled training datasets to generate different models and combine them. Boosting used a weighted training dataset using an iterative process. Boosting increases the weights of previously misclassified instances to encourage the next model to focus on these hard to classify instances. Finally, CostBoosting presented a cost-sensitive adaption of Boosting.

References

[1] J. Baker. Reducing bias and inefficiency in the selection algorithm. *Proc. Second International Conference on Genetic Algorithms*, pages 14–21, 1987.

[2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] L. Breiman. The heuristics of instability in model selection. *Machine Learning*, (24):2350–2383, 1996.

[4] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[5] C. Brodley and M. Friedl. Identifying and eliminating mislabeled training instances. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 799–805, 1996.

[6] E. K. C. Blake and C. J. Merz. Uci repository of machine learning databases. irvine:university of california, department of information and computer sciences, 1998.

[7] H. Drucker and C. Cortes. Boosting decision trees. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 479–485. The MIT Press, 1996.

[8] Freund and Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EUROCOLT: EUROCOLT, European Conference on Computational Learning Theory, EuroCOLT,*. LNCS, 1994.

[9] Y. Freund and R. Schapire. A short introduction to boosting. *J. Japan. Soc. for Artif. Intel.*, pages 771–780, 1999.

[10] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[11] M. H. H. Vafaie, D. Abbott and I. P. Matkovsky. Combining models across algorithms and samples for improved results. In *The Twelfth International Conference on Tools with Artificial Intelligence*, page 344, 351.

[12] J. Quinlan. Boosting and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730. Cambridge, MA. AAAI Press/MIT Press, 1996.

[13] J. R. Quinlan. Bagging, boosting, and c4.5. In *AAAI/IAAI, Vol. 1*, pages 725–730, 1996.

[14] R. E. Schapire. Theoretical views of boosting. In *Proc. 4th European Conference on Computational Learning Theory*, volume 1572, pages 1–10. Springer-Verlag, 1999.

[15] K. M. Ting and Z. Zheng. Boosting cost-sensitive trees. In *Discovery Science*, pages 244–255, 1998.

[16] K. M. Ting and Z. Zheng. Boosting trees for cost-sensitive classifications. In *European Conference on Machine Learning*, pages 190–195, 1998.

[17] I. Trans, I. pp, and Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization, 1998.

[18] A. Tsymbal and S. Puuronen. Bagging and boosting with dynamic integration of classifiers. In *PDKK*, pages 116–125, 2000.

[19] S. M. Weiss, C. Apté, F. J. Damerau, D. E. Johnson, F. J. Oles, T. Goetz, and T. Hampp. Maximizing text-mining performance. *IEEE Intelligent Systems*, 14(4):63–69, 1999.

[20] I. H. Witten and S. E. Frank. *Data Mining.* Morgan Kaufmann Publishers, 2000.