Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

# Dynamic Early Stopping for Naive Bayes

**Aäron Verachtert, Hendrik Blockeel,** and **Jesse Davis**
Department of Computer Science
KU Leuven
Celestijnenlaan 200A 3001 Leuven, Belgium
{aaron.verachtert, hendrik.blockeel, jesse.davis}@cs.kuleuven.be

## Abstract

Energy efficiency is a concern for any software running on mobile devices. As such software employs machine-learned models to make predictions, this motivates research on efficiently executable models. In this paper, we propose a variant of the widely used Naive Bayes (NB) learner that yields a more efficient predictive model. In contrast to standard NB, where the learned model inspects all features to come to a decision, or NB with feature selection, where the model uses a fixed subset of the features, our model dynamically determines, on a case-by-case basis, when to stop inspecting features. We show that our approach is often much more efficient than the current state of the art, without loss of accuracy.

## 1 Introduction

Portable battery-powered devices such as tablets, smartphones, cameras, and intelligent sensors are ubiquitous in our world. With this comes an increasing need to deploy machine-learned models on such systems (see, e.g., [Badami *et al.*, 2016]). However, these devices have constraints on their resources such as battery life, CPU power, memory and disk space. In particular, the CPU time needed to make a prediction with such models directly affects the energy-efficiency of the device, and hence its autonomy. For this reason, studying how to design models that make predictions more efficiently has begun to attract attention [Seldin *et al.*, 2013; Herbrich *et al.*, 2015].

In this paper, we study this question for Naive Bayes. NB is a widely used learning method, in part because the learning phase is efficient and the learned model is often very accurate. However, in the prediction phase, NB can be relatively inefficient, as it inspects all features of the query instance before making a decision. This can be alleviated by performing feature selection: during training, a static subset of the features is selected such that the model based on this subset performs as well as a model based on all features.

We propose a novel model format and learner. Our model format incorporates certain "stopping conditions" into the basic NB model in order to determine dynamically, for each individual query instance, when to stop inspecting more features. This "dynamic stopping" approach can be combined with standard feature selection.

The basic idea is the following. As the NB model iterates over the features, it updates a score for each class; the class that scores highest in the end will be the one predicted. In some cases, one class may be so far ahead of the others that it seems unlikely that any other class will catch up. At that point, we can stop and predict the class that is currently ahead. Such an approach requires a very efficient stopping criterion, otherwise the computational effort of evaluating the criterion may outweigh the gain of not having to evaluate more features. We propose algorithms for determining when and where to include the stopping conditions and how to perform prediction with these models. Empirically, we find that this approach yields substantial efficiency gains compared to both static feature selection and several other baselines without loss of predictive performance.

## 2 Naive Bayes

Assume we want to learn a classifier that predicts the class $Y$ from input variables $X_1, \ldots, X_m$. For ease of explanation, we assume nominal variables, and we systematically use $y$ and $x_i$ to denote a particular value from the domain of $Y$ and $X_i$. The Naive Bayes classifier predicts the class $y$ that maximizes $p(y)\prod_{i=1}^{m} p(x_i|y)$. This prediction is Bayes-optimal if the $X_i$ are conditionally independent given the class; if not, the classifier still works well in practice [Domingos and Pazzani, 1997].

Learning the Naive Bayes classifier is as simple as estimating $p(y)$ and $p(x_i|y)$ for all values of $i$, $x_i$ and $y$, and storing these estimates in a table. The estimates are usually based on frequency counts with Laplace smoothing. In practice, implementations often compute a sum of logs instead of this product, to avoid numerical underflow. That is, with $l(\cdot) = \log \hat{p}(\cdot)$ and $\hat{p}(\cdot)$ the estimate of $p(\cdot)$, practical implementations store the $l$ values in a table during learning, and for prediction return $\arg\max_y l(y) + \sum_{i=1}^{m} l(x_i|y)$.

For binary prediction, where $Y$ can only have values 0 or 1, one can equivalently store for each value of each attribute the difference $d(x_i) = l(x_i|1) - l(x_i|0)$ and predict class 1 if and only if $l(1) - l(0) + \sum_{i}^{m} d(x_i) > 0$. In this case, it is known that 0 is not always the optimal threshold, and it may be better to treat this threshold as a tunable parameter $t$ [Thai-Nghe *et* *al.*, 2010]. The decision criterion then becomes: predict class 1 if and only if

$$l(1) - l(0) + \sum_{i=1}^{m} d(x_i) > t. \tag{1}$$

From here on, we focus on this implementation of Naive Bayes.

One way to speed up Naive Bayes is to not include all $m$ attributes in the model, but only the $k$ most informative ones. These can be chosen using standard feature selection methods, which typically use some function to score attributes according to expected informativeness and then select the $k$ highest-scoring ones.

Another way to speed up Naive Bayes is to stop inspecting attributes as soon as it is clear that the outcome can no longer change. For the positive class, this occurs if the accumulated sum through attribute $i$ is so far above $t$ that it is impossible for the remaining attributes to bring it down to below $t$; that is

$$l(1) - l(0) + \sum_{j=1}^{i} d(x_j) - t > |\sum_{j=i+1}^{m} \min_{x_i \in Dom(X_i)} d(x_i)|.$$

In other words, even if all the unobserved values maximally favor the negative class ($d(x_i)$ is maximally negative), the end result cannot be lower than $t$. A similar rule holds for the negative class when the accumulated sum is too far below $t$.

While this criterion is safe (stopping never gives a different outcome from continuing), it is unlikely to bring any efficiency gain. The stopping condition assumes a worst-case scenario for the unseen attributes. In the (unrealistic) best-case scenario where the first attributes maximally favor class 1, one still expects the condition to be fulfilled only when $i$ is approximately $m/2$. In practice, $i$ will be much closer to $m$. Thus, the typical savings would be less than half of the iterations, while the stopping condition makes each iteration more than twice as expensive.

## 3 Related Work

The same-decision probability is a measure to evaluate the probability that a prediction using only a subset of the available attributes will change after evaluating the remaining attributes [Choi *et al.*, 2012]. For Naive Bayes, this probability can either be approximated or computed exactly [Chen *et al.*, 2014]. By ordering the attributes by informativeness, the assumptions of the approximation techniques are violated which could adversely affect the same-decision probability estimate. In the worst case, computing an exact estimate is exponential in the number of remaining attributes. In contrast, our approach can be viewed as a simple heuristic that gains efficiency and seems to work well in practice but does not offer guarantees.

Another area of related work is cost-sensitive learning (e.g., [Elkan, 2001; Chai *et al.*, 2004; Xu *et al.*, 2013; 2014]). Typically, these approaches focus on whether the cost associated with acquiring an attribute value outweighs the expected improvement in performance. Our problem can be translated into a cost-sensitive setting by defining the cost of each attribute as the CPU time needed to incorporate its value into the prediction. Chai et al. (2004) focus on costs associated with acquiring and using attribute values at test time. Their dynamic strategy for selecting which attributes to acquire is computationally expensive as it evaluates at test time what the benefit of each unobserved feature would be. A more recent cost-sensitive approach is model cascades, which uses a sequence of progressively more expensive models to classify an example. The process stops once there is sufficient confidence in the prediction [Xu *et al.*, 2013; 2014].

Both the same-decision probability and cost-sensitive Naive Bayes can be used to dynamically stop prediction. However, they both rely on performing calculations at prediction time to decide when to stop evaluating extra attributes, which could be computationally demanding. We propose a more efficient method in the next section.

## 4 Dynamic Early Stopping during Naive Bayes Prediction

In standard Naive Bayes, the learned model is a table that contains the $d(\cdot)$ values. Prediction with this model involves iterating over all included attributes, and updating a score for each class after observing the value of each attribute. The final prediction is made only after considering all attributes. In contrast, our proposed approach dynamically decides at test time how many attributes to consider when classifying each example.

### 4.1 Predicting with Stop Points

**Algorithm 1** Prediction with Stop Points

```
Input:
    $d$: table with log-coefficients of the NB model;
    $S$: list of stop points $(k, u, l)$;
    $t$: decision threshold;
    $\mathbf{x} = (x_1, \ldots, x_m)$: instance to be classified
Output: prediction for $\mathbf{x}$

    $\Delta = d[0,0]$
    $start = 1$
    for all $(k, u, l)$ in $S$, from low to high $k$, do
        $\Delta = \Delta + \sum_{i=start}^{k} d[i, x_i]$
        if $\Delta > u$ then return 1
        else if $\Delta < l$ then return 0
        $start = k + 1$
    $\Delta = \Delta + \sum_{i=start}^{m} d[i, x_i]$
    if $\Delta > t$ then return 1
    else return 0
```

The key idea behind our approach is that instead of iterating over all features, the prediction can stop after the $k^{th}$ feature, if a certain stopping condition is fulfilled. To achieve this, we extend the Naive Bayes model with a list of *stop points*. Formally, a stop point is a tuple $(k, u, l)$ where $k$ is an attribute index, and $u$ and $l$ are upper and lower thresholds on an example's predicted probability of being positive after observing the first $k$ attributes.

When making a prediction in a model with stop points, inference iterates over the attributes until it reaches a stop point. Then if $p(y = 1|x_1, \ldots, x_k) > u$ or $p(y = 1|x_1, \ldots, x_k) < l$, inference is stopped and the current prediction is returned. Algorithm 1 shows pseudocode for prediction with this extended model. It assumes $d[i, j]$ with $i > 0$ contains the $d(\cdot)$ value for value $j$ of attribute $i$, and $d[0, 0]$ contains $l(1) - l(0)$ (see Equation 1); $u$ and $l$ here denote the bounds in log space.

## 4.2 Identifying Stop Points

Incorporating stop points into the model complicates the training algorithm. In addition to learning the table (like standard Naive Bayes), it also needs to learn the list of stop points $(k, u, l)$. This entails deciding where to include stop points ($k$) and picking the stopping thresholds $u$ and $l$ for that position. As checking whether inference can be stopped includes a non-negligible computational cost, associating a stop point with each attribute would likely outweigh any potential efficiency gains. Therefore, we need to carefully consider whether it is worthwhile to add a stop point.

Our training algorithm will include a stop point $(k, u, l)$ only if two criteria are fulfilled:

1. **The support criterion:** a sufficient proportion of the instances reaching that stop point is expected to fulfill the stopping condition, and
2. **The accuracy criterion:** the expected accuracy on these instances, if they are assigned the currently higher-scoring class, is high enough.

Note that both criteria rely on expected values. These are estimated using a separate validation set.

In our implementation, the support criterion simply states that the estimated proportion of instances that fulfill the stopping condition (we call these "stopped instances") must be above a given threshold $s$, which is a parameter of the training algorithm.

The accuracy criterion is more complicated. The most intuitive criterion is the following ($R_1$): *For the stopped instances, the expected accuracy if they are assigned the currently highest-scoring class is at least as high as the expected accuracy if they would be assigned the highest-scoring class after seeing all features.* In other words, early stopping does not hurt accuracy. We found this rule not to work well, apparently because it suffers from overfitting. This accounts for possible overestimation of the early-stopping accuracy. In addition, a second rule was added ($R_2$): *The expected accuracy on stopped instances should be $p\%$ higher than the full model's accuracy on all instances*, where $p$ is a parameter. This rule helps avoid that instances are stopped, not because their early-stopping accuracy is high, but because the accuracy of the full model happens to be low. Here, $p$ is an additional countermeasure against overfitting. The accuracy criterion is the conjunction of $R_1$ and $R_2$.

The training algorithm determines stop points as follows. For each position $k$, it checks if a $u$ and $l$ can be found such that $(k, u, l)$ meets both the support criterion and the accuracy criterion. The thresholds need to be set separately after each attribute because NB's probability estimates tend towards 0 or 1 as more attributes are observed. When multiple $u$ and $l$ are found, it chooses the lowest $u$ and highest $l$, as this maximizes the proportion of instances that are stopped.

**Algorithm 2** Training Naive Bayes with Stop Points

```
Input:
  $T$: training set
  $V$: validation set
  $M$: ordering metric
  $s$: support parameter
  $p$: performance threshold
Output:
  $d$: Naive Bayes model
  $S$: list of stop points $(k, u, l)$, where $k$ is the position, $u$ is the upper bound, and $l$ is the lower bound
  $t$: decision threshold for instances that observe all attributes

  Order $\mathbf{X}$ from most to least informative using $M$
  $d$ = TRAINNB($T$)          // Train NB model with all attributes
  $d_i$ = function mapping $x$ to $d$'s score for $x$ after the first $i$ attributes
  $V'$ = $V$   // $V'$ is the set of examples for which inference has not been stopped
  for $i = 1$ to $m$ do
      // Find candidate bounds that meet the accuracy requirements
      $C$ = IDENTIFYCANDIDATETHRESHOLDS
      $U = \{v \in C | \text{MEETSACCREQ}(\{x \in V' | d_i(x) > v\})\}$
      $L = \{v \in C | \text{MEETSACCREQ}(\{x \in V' | d_i(x) < v\})\}$
      // Choose those that eliminate the most examples
      if $U$ is empty then $u = \infty$ else $u = \min(U)$
      if $L$ is empty then $l = -\infty$ else $l = \max(L)$
      // If the support is high enough, insert stop point
      Out = $\{x \in V' | d_i(x) > u \lor d_i(x) < l\}$
      if $|Out| \geq s|V'|$ then
          add $(i, u, l)$ to $S$
          $V'$ = $V' - Out$
  $t$ = PICKDECISIONTHRESHOLD($V'$)

  function MEETSACCREQ($S$)
      return $\text{Acc}(d_i, S) > (1 + p)\text{Acc}(d_m, V)$ & $\text{Acc}(d_i, S) > \text{Acc}(d_m, S)$
```

**Putting it all together** Algorithm 2 provides pseudocode for the training algorithm. As input, it receives a training set $T$, a validation set $V$, an ordering metric $M$, a support threshold $s$, and an improvement threshold $p$. Using the training data $T$, a Naive Bayes classifier is trained using all the attributes. It statically orders the attributes according to $M$ such that the most informative ones are evaluated first in an attempt to arrive at a decision as quickly as possible.

Next, the algorithm iterates over the attributes from most to least informative and considers introducing a stop point after each attribute $k$. It treats the predicted probability of each example in the validation data as a candidate upper (lower)

| Name | #I | #A | #p / #n |
|---|---|---|---|
| RCV1-v2 | 804,414 | 47,236 | 381,327 / 423,087 |
| KDD 2009 | 50,000 | 15,000 | 3,674 / 463,126 |
| p53 | 31,422 | 5,408 | 152 / 31,270 |
| Thrombin | 2,545 | 139,351 | 193 / 2,352 |
| YouTube | 122,046 | 1,000 | 31,722 / 90,324 |
| Gisette | 7,000 | 5,000 | 3,500 / 3,500 |
| IMDb.drama | 120,670 | 1,001 | 43,679 / 76,991 |

Table 1: Summary of the used data sets. For each data set, the number of instances (#I), the number of attributes (#A) and the class distribution (#p / #n) is listed.

threshold[1] and identifies all thresholds such that classifying all examples in the validation data with a predicted probability above (below) the threshold as positive (negative) satisfies the two accuracy conditions. Of the thresholds that meet the accuracy requirements, it then picks the upper and lower thresholds that apply to the largest number of examples. If the two thresholds apply to at least $s|V|$ examples in aggregate, a stop point $(k, u, l)$ is added. After adding a stop point, all stopped examples are removed from the validation data.

## 5 Experimental Evaluation

The goal of our empirical evaluation is to answer the following questions:

1. For a fixed attribute budget $k$, how does the dynamic approach compare to the static one in terms of efficiency and accuracy?
2. How does our proposed approach compare to same-decision probability (SDP) [Choi *et al.*, 2012] and cost-sensitive Naive Bayes (csNB) [Chai *et al.*, 2004]? These represent two other ways to dynamically make a prediction based on only a subset of the attributes.

To answer the first question, we train two types of models for each attribute ordering. The baseline is a "static" model that always uses $k$ attributes to make a prediction. Then we learn a dynamic model according to our proposed approach that can consider at most $k$ attributes at prediction time. To explore a range of different operating conditions, we investigate various values of $k$.

### 5.1 Data Sets

We perform an evaluation using seven data sets from various domains, summarized in Table 1. Each data set has at least 1000 attributes. We treat each task as a binary classification problem and we discretize numerical features by computing the mean and standard deviation for each of these features on the training data. We construct four bins. The first bin contains 0, as most data sets are very sparse. The three remaining bins are: $(-\infty, (\mu - \sigma)]$, $((\mu - \sigma), (\mu + \sigma)]$, and $((\mu + \sigma), +\infty)$.

[1]This is analogous to the standard approach to discretizing a continuous feature in decision tree learning or building an ROC curve.

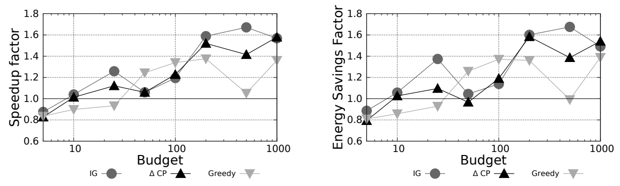

Figure 1: Time (left) and energy (right) efficiency of the dynamic approach, relative to the static one, measured on the Raspberry Pi for the IMDb.drama data set.

### 5.2 Methodology

This section discusses the methodology of our empirical evaluation. First, we discuss the experimental setup, and then we explain which evaluation metrics we use.

**Experimental Details**

We consider the following values for the budget $k$: 5, 10, 25, 50, 100, 200, 500, 1000. For our approach, we set $p = 0.05$ and $s = 0.05$ and did not try other values. For attribute orderings, we consider a greedy, hill-climbing wrapper-based feature selection method (Greedy) [Kohavi and John, 1997]; FCBF [Yu and Liu, 2003], a correlation-based feature selection method; information gain (IG) and chi square (Chi), which are standard filter-based feature selection criteria; and a novel metric tailored to Naive Bayes that estimates the difference in conditional probabilities (ΔCP), which is defined as:

$$\Delta CP(X_i) = \sum_{x_i \in dom(X_i)} \hat{p}(x_i)|\hat{p}(x_i|1) - \hat{p}(x_i|0)|. \quad (2)$$

Intuitively, this is a weighted average of how discriminative a particular attribute is, given that Naive Bayes uses the conditional probabilities to build up its "confidence" in a given prediction.

Each data set is split into a training set, a validation set and a test set. As RCV1-v2 is usually treated as a temporal data set, we use the standard chronological split [Lewis *et al.*, 2004], where the training set consists of the first 23,149 instances, the validation set consists of the next 23,149 instances, and the test set consists of the remaining instances. For the other data sets, we randomly select 40% of the examples for training, 20% for validation, and 40% for testing. We repeat this procedure ten times and report an average across all runs.

**Evaluation Metrics**

We want to measure two properties of the learned models: efficiency and predictive performance. For efficiency, we begin by using a Raspberry Pi system (Model B+, 512MB, with Power Bank battery pack supplying 5V at a maximum of 1A) running Raspbian and a Java Virtual Machine, and measure energy consumption using a Tenma digital multimeter 72-7730A. Energy consumption in kWh is calculated by multiplying the time it takes to make a prediction for all examples in the test set by the average electric current during prediction and the steady voltage of 5V.

Taking energy measurements with such a system is better than using a device like a smartphone, where it is harder to

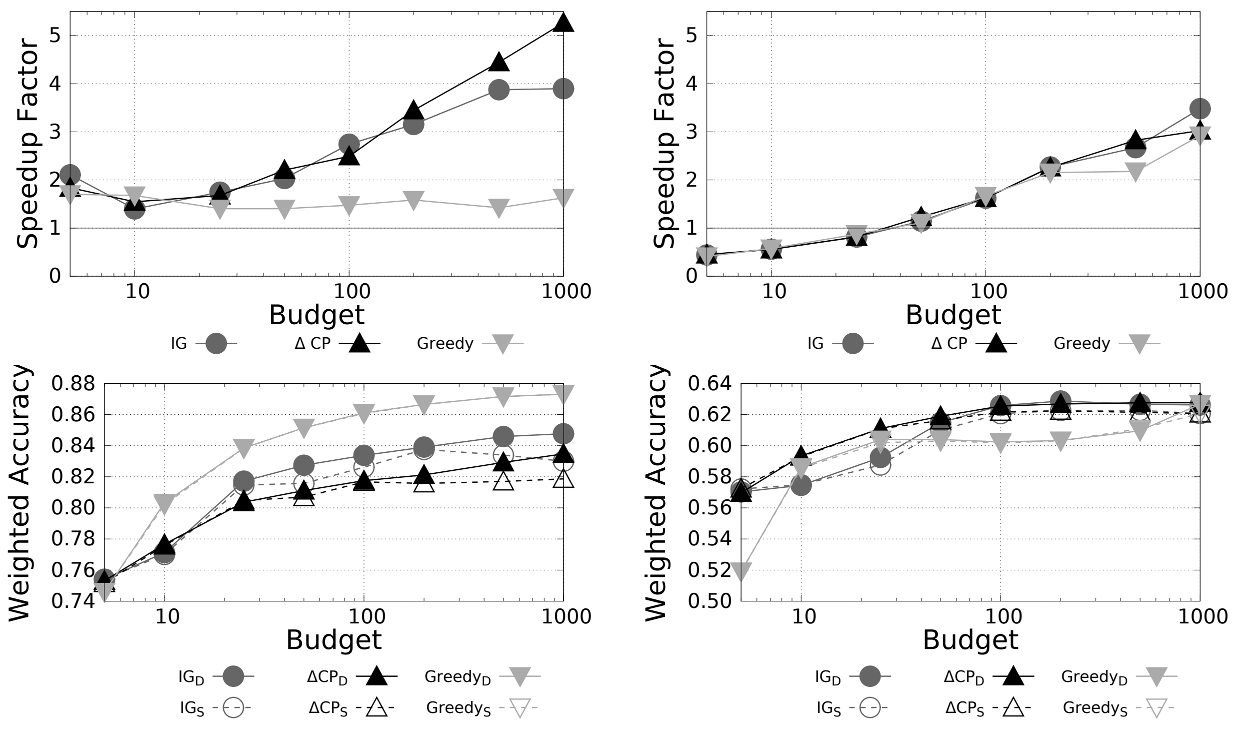

Figure 2: Speedup factor and weighted accuracy as a function of attribute budget, for the RCV1-v2 (left) and YouTube (right) data sets. For readability, we only show the curves for three attribute rankings: IG, $\Delta$CP, Greedy. For the weighted accuracy plots, the dynamic models ($\cdot_D$) are shown in solid lines and the static models ($\cdot_S$) in dashed lines. On the weighted accuracy plots, the lines for the corresponding static and dynamic approaches are largely overlapping, indicating that using the dynamic approach does not result in a loss of predictive performance. For example, on the RCV1-v2 data set, the curves for the dynamic and static variants using Greedy attribute selection are completely overlapping.

control which other applications are running. Due the constrained environment, this type of evaluation is very time consuming. It has been observed elsewhere [Verachtert *et al.*, 2015] that CPU time correlates well with energy usage. After double-checking this claim, we therefore use CPU time for prediction as a proxy for energy usage.

Because several of our data sets have an imbalanced class distribution, simply reporting accuracy would lead to several cases where the majority classifier would achieve very good performance. Therefore, we use weighted accuracy as our metric, with a misclassification cost of 1 for a negative example and $f(0)/f(1)$ for a positive example, where $f(y)$ is the frequency of class $y$ in the validation set. We estimate the costs based on the class distribution in the validation data. The same weighted accuracy metric is used for the accuracy requirements in the training procedure. Note that using a ranking metric such as area under the ROC curve would be challenging for the dynamic approach as the number of attributes it considers depends on the example. Thus, the probability estimates may not be comparable across stop points.

## 5.3 Results and Discussion

Figure 1 shows the result of an experiment with the Raspberry Pi, using the IMDb.drama data set. The x-axis shows the budget, the y-axis the speedup factor (left) or relative energy efficiency (right) for the dynamic approach, relative to the static approach. This graph confirms two things: (a) on the Raspberry Pi, our approach can lead to significant energy savings; and (b) speedup is a good proxy for energy efficiency (the difference between these two is on average 0.03 in these data, with one outlier of 0.11 at budget 25). Because CPU time is much easier to measure, we henceforth focus on speedup factors, measured on desktops.

Figure 2 shows the speedup factor and the difference in weighted accuracy as a function of attribute budget for two representative data sets: YouTube and RCV1-v2.[2] For RCV1-v2, regardless of the ordering heuristic, the dynamic approach always results in faster prediction times than the static approach. In terms of accuracy, the curves for the static and dynamic approach for each ordering metric are largely overlapping, indicating that the runtime improvements come with no loss in predictive performance. For YouTube, the prediction times for the dynamic approach are slightly slower for small values of $k$, due to the extra cost associated with filtering out examples. However, for values of $k > 25$ the dynamic approach results in substantial savings in prediction time. Furthermore, these are the values of $k$ that result in the

[2]Full experimental results are available in the online supplement at http://dtai.cs.kuleuven.be/software/dsnb

|  | Chi |  | IG |  | Δ CP |  | Greedy |  | FCBF |  |
|---|---|---|---|---|---|---|---|---|---|---|
|  | SF | ΔwAcc | SF | ΔwAcc | SF | ΔwAcc | SF | ΔwAcc | SF | ΔwAcc |
| RCV1-v2 | 2.55 | +0.44 | 3.15 | +0.15 | 5.27 | +1.60 | 1.63 | +0.00 | 1.37 | -1.18 |
| KDD 2009 | 1.99 | +0.04 | 1.11 | -0.06 | 1.25 | +0.56 | 1.53 | -0.10 | 1.47 | -0.34 |
| p53 | 3.40 | -1.36 | 2.83 | +0.07 | 3.03 | -0.57 | 1.89 | -0.59 | 2.16 | -3.51 |
| Thrombin | 1.60 | -0.23 | 1.02 | -0.40 | 4.88 | -0.91 | 0.97 | -0.22 | 1.13 | -4.91 |
| YouTube | 1.93 | +0.61 | 2.67 | +0.40 | 2.26 | +0.44 | 2.92 | +0.60 | 2.26 | -0.26 |
| Gisette | 4.43 | +0.49 | 4.42 | +0.71 | 4.12 | +0.82 | 1.91 | +0.02 | 1.38 | -0.29 |
| IMDb.drama | 4.43 | +0.49 | 4.42 | +0.71 | 4.12 | +0.82 | 1.91 | +0.02 | 1.38 | -0.29 |

Table 2: For the budget that maximizes the validation set accuracy of the static model (separately optimized for each data set), the table lists the speedup factor (SF) and gain in weighted accuracy (ΔwAcc) of the dynamic model.

best predictive performance. Again, we observe that using the dynamic approach results in nearly equivalent predictive performance as the static approach.

Table 2 compares the static model with the attribute budget that maximizes the validation set weighted accuracy to the dynamic model with the same budget for each feature ranking method in terms of speedup factor (SF) and difference in weighted accuracy (ΔwAcc). Across the 35 settings, the dynamic approach results in an improved runtime 34 times. Speedup factors vary from 0.97 to 5.27. For ΔwAcc, positive numbers indicate that the dynamic approach is better than the static one. The dynamic and static models tend to have similar weighted accuracy scores: sometimes one is better, sometimes the other, with the difference typically below 1%. In summary, the dynamic approach achieves substantial runtime gains without sacrificing predictive performance.

Our approach makes these energy improvements while also respecting the memory restrictions of a resource-constrained environment. Augmenting the model with the stop points requires storing one integer and two floating point numbers per stop point. In practice, this corresponds to an increased memory use (averaged over our data sets) of 15% for $k = 10$, 3% for $k = 100$ and 0.4% for $k = 1000$.

Finally, we compared our approach to csNB and SDP. These can also be used to make a decision whether to stop prediction, but require more calculations during the inference process. In our experiments, the computation time csNB needed to decide whether to make a prediction or see more attributes depended linearly on the number of remaining unobserved attributes, and was about 28 times that of simply incorporating all remaining attributes into the prediction. For SDP, we found that it is only faster than evaluating all remaining attributes when less than 10 attributes are left; with 20 attributes remaining, computing the SDP was more than 1000 times slower than simply evaluating the last 20 attributes. Consequently, neither of these approaches offers much of an improvement over static feature selection. In contrast, our approach results in substantial efficiency improvements. Essentially, the runtime cost csNB and SDP impose for dynamically deciding when to stop the prediction will only yield benefits when acquiring an attribute value is extremely expensive either in terms of energy (computation) or monetary cost.

## 6 Conclusion

As deploying machine-learned models becomes more prevalent on devices with constrained resources, it becomes important to reduce the CPU burden associated with making predictions. We tackled this problem for Naive Bayes by employing a dynamic strategy that stops the prediction process for test examples whose label is sufficiently certain. Empirically, we found that our dynamic approach can reduce the energy consumption per example by over 80% compared to a static approach, without significantly affecting accuracy. In the future, we would like to explore how to incorporate feature extraction costs into the model data sets, and measure the energy usage associated with predictions on a sensor.

## Acknowledgments

Aäron Verachtert and Hendrik Blockeel are partially supported by FWO-Vlaanderen (G.0179.10, G0682.11) and KU Leuven Research Fund (CREA/13/014, GOA/13/010). Jesse Davis is partially supported by the KU Leuven Research Fund (C22/15/015), and FWO-Vlaanderen (G.0356.12, SBO-150033). We thank Jan Van Haaren, Wannes Meert, and the anonymous reviewers for their feedback.

## References

[Badami *et al.*, 2016] Komail Badami, Steven Lauwereins, Wannes Meert, and Marian Verhelst. A 90 nm CMOS, 6 $\mu$W power-proportional acoustic sensing frontend for voice activity detection. *IEEE Journal of Solid-State Circuits*, 51(1):291–302, 2016.

[Chai *et al.*, 2004] Xiaoyong Chai, Lin Deng, Qiang Yang, and Charles X. Ling. Test-cost sensitive naive Bayes classification. In *Proceedings of the 4th IEEE International Conference on Data Mining*, pages 51–58, 2004.

[Chen *et al.*, 2014] Suming Jeremiah Chen, Arthur Choi, and Adnan Darwiche. Algorithms and applications for the same-decision probability. *Journal Artificial Intelligence Research*, 49:601–633, 2014.

[Choi *et al.*, 2012] Arthur Choi, Yexiang Xue, and Adnan Darwiche. Same-decision probability: A confidence measure for threshold-based decisions. *International Journal of Approximate Reasoning*, 53(9):1415–1428, 2012.

[Domingos and Pazzani, 1997] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[Elkan, 2001] Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, 2001.

[Herbrich *et al.*, 2015] Ralf Herbrich, Venkatesh Saligrama, Kilian Q. Weinberger, Joe Wang, Tolga Bolukbasi, and Matt J. Kusner, editors. *ICML Workshop on Resource-Efficient Machine Learning*, 2015.

[Kohavi and John, 1997] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[Lewis *et al.*, 2004] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[Seldin *et al.*, 2013] Yevgeny Seldin, Yasin Abbasi-Yadkori, Koby Crammer, Ralf Herbrich, and Peter Bartlett, editors. *NIPS Workshop on Resource-Efficient Machine Learning*, 2013.

[Thai-Nghe *et al.*, 2010] Nguyen Thai-Nghe, DT Nghi, and Lars Schmidt-Thieme. Learning optimal threshold on resampling data to deal with class imbalance. In *Proceedings of IEEE RIVF International Conference on Computing and Telecommunication Technologies*, pages 71–76, 2010.

[Verachtert *et al.*, 2015] Aäron Verachtert, Wannes Meert, Jesse Davis, and Hendrik Blockeel. Empirical study: Energy usage of standard machine learning prediction. In *Proceedings of the ICML 2015 Workshop on Resource-Efficient Machine Learning*, 2015.

[Xu *et al.*, 2013] Zhixiang Xu, Matt Kusner, Minmin Chen, and Kilian Q Weinberger. Cost-sensitive tree of classifiers. In *Proceedings of the Thirtieth International Conference on Machine Learning*, pages 133–141, 2013.

[Xu *et al.*, 2014] Zhixiang Eddie Xu, Matt J. Kusner, Kilian Q. Weinberger, Minmin Chen, and Olivier Chapelle. Classifier cascades and trees for minimizing feature evaluation cost. *Journal of Machine Learning Research*, 15(1):2113–2144, 2014.

[Yu and Liu, 2003] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 856–863, 2003.