# Hierarchy-based Partition Models: Using Classification Hierarchies to Improve the Statistical Estimation of Bigrams

**Matthias Buch-Kromann and Martin Haulrich**
Center for Research in Translation and Translation Technology
Copenhagen Business School
`mbk.isv@cbs.dk, mwh.isv@cbs.dk`

Working paper

## Abstract

We propose a novel machine learning technique that can be used to estimate probability distributions for categorical random variables that are equipped with a natural set of classification hierarchies, such as words equipped with word class hierarchies, wordnet hierarchies, and suffix and affix hierarchies. We evaluate the estimator on bigram language modelling with a hierarchy based on word suffixes, using English, Danish, and Finnish data from the Europarl corpus with training sets of up to 1–1.5 million words. The results show that the proposed estimator outperforms modified Kneser-Ney smoothing in terms of perplexity on unseen data. This suggests that important information is hidden in the classification hierarchies that we routinely use in computational linguistics, but that we are unable to utilize this information fully because our current statistical techniques are either based on simple counting models or designed for sample spaces with a distance metric, rather than sample spaces with a non-metric topology given by a classification hierarchy.

*Keywords:* machine learning; categorical variables; classification hierarchies; language modelling; statistical estimation

## 1 Introduction

The notion of smoothing plays a central role in most statistical approaches to natural language processing, and many of the most important techniques in statistics and machine learning can be viewed as instances of smoothing. Smoothing is based on the idea that if we observe an event, then this should increase our belief in observing this particular event and similar events again in the future. The different smoothing techniques mainly differ with respect to their underlying notion of similarity. When the sample space has a natural interpretation as a metric space, similarity is naturally defined in terms of metric distance: points in the sample space are similar if they are close to each other. This distance-based notion of similarity is the starting point for a wide array of statistical estimation methods, including generalized linear models, support vector machines, kernel smoothing, and $k$-nearest neighbour methods (see Hastie et al. (2009) for an overview). The options are more limited when the sample space does not have a natural metric, eg, when the data are categorical rather than real-valued. One standard technique is to map the categorical sample space to a high-dimensional metric space by means of feature functions, thereby imposing a metric on the categorical space which allows us to apply generalized linear models, support vector machines, and other distance-based estimators to the categorical data. Although this often produces excellent results, each feature corresponds to a separate dimension, and the high dimensionality of the resulting metric space can make it difficult to model higher-order interactions between the features; moreover, the topology of the resulting metric space may not necessarily correspond to any intuitively plausible notion of similarity in the untransformed categorical space. Feature-based techniques do therefore not always produce optimal results.

Another standard technique is to apply simple counting models to the untransformed data, eg, by ordering the dimensions in the categorical sample space with respect to their importance, and smoothing higher-order models with lower-order models that ignore the least important dimensions. This approach is particularly suited to problems like $n$-gram language modelling where the sample space is extremely large and the interactions between the dimensions are highly non-linear. Interpolated $n$-gram models such as Kneser-Ney smoothing (Kneser and Ney, 1995) are instances of this approach (see Chen and Goodman (1998) and Goodman (2001) for a comprehensive overview of state-of-the-art language modelling techniques).

In this paper, we will present an alternative density estimation technique extending a proposal by Buch-Kromann (2006) that can be used to estimate joint and conditional probabilities for categorical random variables by correcting an initial estimator. The method resembles count-based methods by working on the untransformed categorical data, but uses classification hierarchies to provide a more fine-grained notion of similarity than $n$-gram hierarchies or the metrics produced by feature transformations. We evaluate our estimator on bigram language modelling for English, Danish, and Finnish with a primitive suffix hierarchy, and show that the estimator performs well compared to Kneser-Ney smoothing, ie, even a primitive suffix hierarchy can provide information that can be used to construct improved estimators. Finally, we will discuss the potential ramifications of classification hierarchies in statistical natural language processing in general.

## 2 Classification Hierarchies

Classification hierarchies arise naturally in a wide range of categorical data where it is difficult to define a natural notion of a distance metric. Some prominent examples include the taxonomies used in biological classification, the universal classification systems used by libraries to classify books, and the industrial classification systems used by governments to classify industries. In natural language processing, examples of classification hierarchies for words include word class hierarchies, inflectional hierarchies, and ontological hierarchies such as WordNet (Fellbaum, 1998); word suffixes and word prefixes can be used to define natural hierarchies for words as well. In this article, we will focus exclusively on suffix hierarchies for bigram language modelling in statistical natural language processing, but the usefulness of our proposed estimation method is not restricted to bigram language modelling or statistical natural language processing, but applies readily to other kinds of hierarchies and other problem areas as well. Our main reason for focusing on suffix hierarchies in this paper is that they do not provide any information beyond the information that is encoded in the text itself, without any additional lexical resources, and therefore allow for a more "pure" model comparison with standard methods in statistical language modelling, such as Kneser-Ney smoothing.

Although suffixes are not rich from a linguistic point of view, they are used to encode morphological markers in many languages, and do allow us to capture generalizations that cannot be captured by means of full word sequences. For example, if we use suffix classes for smoothing, we can potentially predict that the sequence "Monday evening" has a higher probability than the sequence "evening Monday", even if both sequences are unseen in the training data. This can happen if the training data contain instances of "Tuesday morning" and "Friday evening," which would result in an increased probability for sequences of the form "•day •ning", where "•" represents an arbitrary character sequence within a single token. In contrast, word-based smoothing methods such as Kneser-Ney would treat these sequences as completely unseen, fall back on unigram probabilities, and predict the same probability for both sequences. Figure 1 shows a partial suffix hierarchy for bigrams, with examples of matching bigrams at terminal nodes.

In formal terms, we define a *classification hierarchy* on a sample space $\Omega$ as any collection $H$ of subsets of $\Omega$ which is closed under intersection and contains $\Omega$ and $\emptyset$. We will refer to the elements of $H$ as *classes*, and we will assume that every class in $H$ is given a unique name that denotes the class — in particular, the top class $\Omega$ is denoted by $\top$, and the empty class $\emptyset$ is denoted by $\bot$. The maximal proper subclasses of a class are called its *immediate subclasses*, and the *depth* of a class is defined as the length of the shortest path from $\top$ to the class via immediate subclasses. It is an important property of these hierarchies that we can construct new hierarchies from existing hierarchies by means of Cartesian products: ie, given any two hierarchies $H, H'$ for $\Omega, \Omega'$, the product set $H \times H'$ defines a natural *product hierarchy* on $\Omega \times \Omega'$.

## 3 Hierarchy-based Partition Models

We can now state our estimation problem in formal terms and define our notion of Hierarchy-

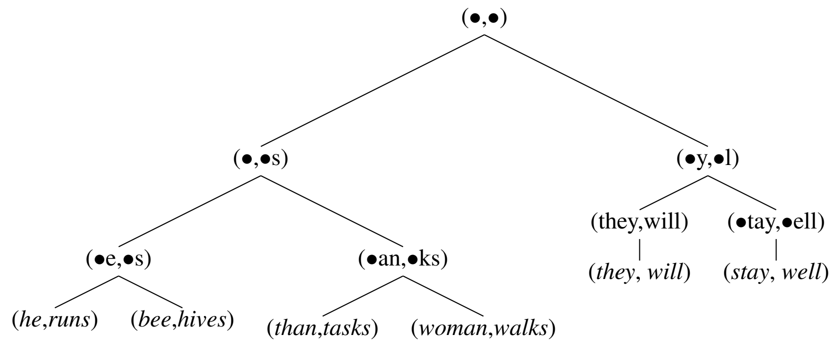


Figure 1: A partial suffix hierarchy for bigrams.

based Partition Models (HPMs). Let $X$ be a random variable with sample space $\Omega_X$ and true distribution $F_X$.[1] Our goal is to compute an estimate $\hat{F}$ of $F_X$ given a set $\bar{x} = (x_1, \ldots, x_n)$ of i.i.d. observations of $X$, a *background distribution* $\tilde{F}$ that provides an initial estimate of $F_X$, and a classification hierarchy $H_X$ whose classes are measurable with respect to $\tilde{F}$ (ie, they have a well-defined probability).

The intuition behind our setup is that we compute $\hat{F}$ as a correction estimator that examines whether there are regions of $\Omega_X$ (as defined by the hierarchy $H_X$) that contain more observations than would be expected with the background distribution $\tilde{F}$; if so, the probability mass assigned to these regions will be adjusted upwards, and the probability mass assigned to the remaining regions will be adjusted correspondingly downwards. In hierarchy-based partition models, we attempt to circumvent the sparse data problem by adjusting the probability of entire regions rather than individual sample points, and falling back on the background estimator in order to estimate the probability of the individual sample points within a region. The background estimator functions as our initial estimator, and is typically based on a simpler model with smaller variance but larger bias which assumes independence between some of the dimensions in $\Omega_X$, and therefore fails to model their interactions.

The intuition behind our correction estimator is illustrated in Figure 2, which shows how the density estimate is improved by incrementally subdividing the sample space into smaller regions. Initially, the sample space is viewed as a single region by our estimator, and the background distribution is used without modification (step 0). In step 1, we cut out a region $c_1$ with a higher probability than expected under the background distribution, and similarly with the regions $c_{11}$ and $c_{12}$ in steps 2 and 3. The remaining part of $c_1$ after cutting out $c_{11}$ and $c_{12}$ now fails to have significantly higher probability than the rest of the original sample space $\top$, and we therefore merge it with $\top$ by means of a special "ghosting" operation in step 4. In step 5, we identify a new high-probability region $c_2$ of the rest space associated with $\top$, some of which has been cut out by the region $c_{12}$, and adjust the probabilities upwards in $c_2$. The partitioning and ghosting operations are explained in more detail in section 5.

Formally, the regions used for partitioning the sample space in our hierarchy-based partition models are defined by means of ordered covers. An *ordered cover* on $H_X$ is a sequence of classes in $H_X$ of the form $o = (c_1, \ldots, c_n)$ where $c_n$ equals the top class $\top$. The *kth partition* in $o$ corresponds to the subset $\pi_k = c_k - c_1 \cup \ldots \cup c_{k-1}$ of $\Omega_X$. Obviously, the partitions in $o$ are disjoint with union $\Omega_X$. By an abuse of notation, we write $o(x) = k$ if $x \in \pi_k$ and refer to $o(x)$ as the *partition index* of $x$ in $o$; the definition of $\pi_k$ implies that $o(x)$ coincides with the smallest $k$ such that $x \in c_k$.

Given an ordered cover $o$ and a set of partition weights $\lambda$, we formally define the *hierarchy-based partition model* $\hat{F}_{o,\lambda}$ as the distribution given by:

$$\hat{F}_{o,\lambda}(s) := \sum_{k=1}^{n} \lambda_k \tilde{F}(s \cap \pi_k) \qquad (1)$$

where $s$ is any measurable subset of $\Omega_X$, and where the partition weights $\lambda = (\lambda_1, \ldots, \lambda_n)$ are chosen so that $\hat{F}_{o,\lambda}(\top) = 1$ and $\lambda_k \geq 0$ for all $k$. Conditional probabilities can be computed straight-forwardly from the joint distribution by

[1]We formally define a distribution as a probability measure on a measurable space over a sample space $\Omega$, cf. Halmos (1974).

means of:

$$\hat{F}_{o,\lambda}(s|s') = \frac{\hat{F}_{o,\lambda}(s \cap s')}{\hat{F}_{o,\lambda}(s')}. \quad (2)$$

For a specific outcome $x$, we observe that $x$ belongs to the unique partition $o(x)$, so that the probability (1) above reduces to the special case:

$$\hat{F}_{o,\lambda}(x) = \lambda_{o(x)}\tilde{F}(x). \quad (3)$$

The computation of $\hat{F}_{o,\lambda}(x)$ for a single data point $x$ therefore includes two main steps: computing the partition index of $x$, and evaluating the background distribution on $x$. The partition index could be computed by searching the classes in the ordered cover sequentially from left to right. However, for large ordered covers, the computational cost of a linear search through the ordered cover would be prohibitive in practice. Fortunately, we can make the search much more efficient by imposing a tree structure on the partitions in the ordered cover, and if the tree is restricted so that the nodes in the tree have a bounded number of children and the tree is approximately balanced (i.e., the height is $O(\log|o|)$), the time complexity of the search will drop from linear $O(|o|)$ to logarithmic $O(\log|o|)$, a quite significant improvement from a computational point of view.

For this reason, we will define an ordered cover $o$ as an *ordered tree cover* if the classes in $o$ are organized in a tree structure that is compatible with the class hierarchy and the ordered cover, ie, $o$ has root partition $\text{root}(o) = |o| = n$, the $k$th partition has parent partition $\text{par}_k(o) > k$ and ordered children $\text{chi}_k(o)$, the class $\text{c}_k$ is always a subclass of $\text{c}_{\text{par}_k}$, and the linear order of the nodes in $o$ corresponds to a post-order walk through the tree structure. As a notational convenience, we will frequently omit the ordered cover and write for example $\text{par}_k$ rather than $\text{par}_k(o)$ if the ordered cover is clear from the context.

In order to speed up the search through the ordered tree cover even more, we will allow the ordered tree cover to contain *ghost partitions* in which $\text{c}_k = \bigcup_{i\in\text{chi}_k} \text{c}_i$ so that $\pi_k$ equals the empty set (ie, the entire region corresponding to $\text{c}_k$ has been cut out by the child partitions in $\text{chi}_k$). We will assume that ghost partitions have an associated *ghost class* $\text{c}_k^{\text{ghost}}$ which must be a superclass of $\text{c}_k$ and a subclass of $\text{c}_{\text{par}_k}$; for non-ghost partitions, we assume $\text{c}_k^{\text{ghost}} = \text{c}_k$. Although ghost-partitions are empty and therefore do not have

```
Input: tree cover $o$, observation $x$, partition $k$.
Output: partition in subtree $k$ containing $x$,
        or null.

foreach $i \in \text{chi}_k$ do
    if $x \in \text{c}_i^{\text{ghost}}$ then
        match := PartitionIndex($x$, $i$);
        if match $\neq$ null then
            return match;
if $k$ is a ghost partition then
    return null;
else
    return $k$;
```

**Algorithm 1**: PartitionIndex($o$, $x$, $k$)

any effect on the probabilities assigned by the hierarchy-based partition model, a ghost partition speeds up the computation of the partition index for outcome $x$ by providing a convenient way of grouping its children into a subtree which can be pruned away in a single step if $x \notin \text{c}_k^{\text{ghost}}$. The algorithm for computing the partition index with ghost partitions is given in Algorithm 1, with the partition index $\text{PartitionIndex}(o, x)$ for $x$ in $o$ computed as $\text{PartitionIndex}(o, x, |o|)$.

In the rest of the paper, we let $\text{sub}_k(o)$ denote the set of all transitive child partitions in the subtree corresponding to partition $k$, defined recursively by:

$$\text{sub}_k(o) := \text{chi}_k \cup \bigcup_{i\in\text{chi}_k} \text{sub}_i.$$

We let $\text{pre}_k(o) := \{1, \ldots, k-1\}$ denote the set of partitions that precede partition $k$ in $o$, and we let $\text{pre}_k^{\max}(o)$ denote the maximal partitions in $\text{pre}_k$, defined by:

$$\text{pre}_k^{\max} := \text{pre}_k - \bigcup_{i\in\text{pre}_k} \text{sub}_i.$$

## 4 Estimating partition weights

We now turn to the problem of estimating a set of partition weights for a Hierarchy-based Partition Model from a data set, given a background distribution and an ordered cover. Ie, we want to estimate partition weights $\lambda$ for a Hierarchy-Based Partition Model $\hat{F}_{o,\lambda}(X)$ given an ordered cover $o$ and observed data $\bar{x}$. In the following, we let $\bar{x}_k(o) := \bar{x} \cap \pi_k(o)$ denote the observations in $\bar{x}$ that fall within partition $k$, and we let $\bar{x}_k^{\text{tree}}(o) := \bar{x} \cap \pi_k^{\text{tree}}(o)$ denote the set of observations in $\bar{x}$ that fall within subtree $k$.

When estimating partition weights, the simplest solution is to estimate the probability of a partition by the empirical probability that an observation in the data set falls within the partition. That is, we can simply use the *empirical partition weight* $\lambda_k^{\text{emp},\bar{x}}$ for partition $k$ given by:

$$\lambda_k^{\text{emp},\bar{x}} := \frac{|\bar{x}_k|}{|\bar{x}|} \cdot \frac{1}{\tilde{F}(\pi_k)}$$

Unfortunately, this approach can be expected to lead to overfitting when data are sparse and the individual partitions contain only very few observations. To circumvent the problem, we can follow standard methodology and use a smoothed set of partition weights. The smoothing problem has been studied extensively in the context of language modelling (see Chen and Goodman (1998)), and the consistently best results have been obtained with smoothing methods based on absolute discounting, where a fixed pseudocount is subtracted from the observed count and distributed among the unobserved outcomes (Ney et al., 1994). We will therefore produce a smoothed estimate of the partition weights by assuming that subtree $k$ passes on the *absolute discount* $D_k^{\text{tree},\bar{x}}(o) := D(|\bar{x}_k^{\text{tree}}|)$ to its parent subtree, and that partition $k$ contributes with discount $D_k^{\bar{x}}(o) := D(|\bar{x}_k|)$ to subtree $k$, where $D$ is a non-decreasing concave discounting function with $0 \leq D(n) \leq n$. The discounted partition weights $\lambda_k^{\text{disc},\bar{x}}(o)$ are then computed as follows.

Define the *inherited interpolation mass* $\text{inh}_k^{\bar{x}}(o)$ that subtree $k$ inherits from its parent by:

$$\text{inh}_k^{\bar{x}} := \text{int}_{\text{par}_k}^{\bar{x}} \cdot \frac{\tilde{F}(\pi_k^{\text{tree}})}{\tilde{F}(\pi_{\text{par}_k}^{\text{tree}})}$$

Define the *total interpolation mass* $\text{int}_k^{\bar{x}}(o)$ for subtree $k$ as the interpolation mass inherited from the parent tree plus the absolute discounts from its child subtrees and partition $k$, minus the discount that the subtree passes on to its parent subtree, ie, as the quantity:

$$\text{int}_k^{\bar{x}} := \text{inh}_k^{\bar{x}} - D_k^{\text{tree},\bar{x}} + D_k^{\bar{x}} + \sum_{i \in \text{chi}_k} D_i^{\text{tree},\bar{x}}$$

Note that the interpolation mass is always non-negative because the concavity of the discounting function ensures that the sum of the incoming and outgoing discounts is non-negative.

Finally define the *pseudo-count* $\text{pseu}_k^{\bar{x}}(o)$ for partition $k$ by:

$$\text{pseu}_k^{\bar{x}} := |\bar{x}_k| - D_k^{\bar{x}} + \text{int}_k^{\bar{x}} \cdot \frac{\tilde{F}(\pi_k)}{\tilde{F}(\pi_k^{\text{tree}})}$$

Using absolute disconting, we can now define a *discounted partition weight* $\lambda_k^{\text{disc},\bar{x}}(o)$ for non-ghost partition $k$ by:

$$\lambda_k^{\text{disc},\bar{x}} := \frac{\text{pseu}_k^{\bar{x}}}{|\bar{x}|} \cdot \frac{1}{\tilde{F}(\pi_k)} \tag{4}$$

This definition only works for non-ghost partitions where $\tilde{F}(\pi_k)$ is non-zero, so for completeness, we let ghost partitions inherit the partition weight from their parent partition by formally defining their partition weight as $\lambda_k^{\text{disc},\bar{x}} := \lambda_{\text{par}_k}^{\text{disc},\bar{x}}$. Since ghost partitions are always empty, their partition weight does not have any practical significance in the computation of joint and conditional probabilities. But it is convenient to have a partition weight for ghost partitions also, since that will allow us to determine whether a partition represents a *bump* or a *dip*, in the terminology of Friedman and Fisher (1999) — ie, whether it has higher or lower weight than its parent partition. In general, since observations are concentrated in bumps rather than dips, partition weights associated with bumps can be estimated more reliably than partition weights associated with dips, especially if the data are sparse. For this reason, we will say that a partition is *degenerate* if $\lambda_k \leq \lambda_{\text{par}_k}$, and prune away all degenerate partitions in the ordered covers that we produce.

## 5 Model induction

So far, we have only described how to induce a set of smoothed partition weights from a data set given a specific ordered cover. Since it is obviously impossible to specify a good ordered cover a priori, we will now turn to the problem of inducing ordered covers as well. As before, we assume we are given a dataset $\bar{x}$. Since we can compute a set of smoothed partition weights $\lambda(o) := \lambda^{\text{disc},\bar{x}}(o)$ for any ordered cover $o$, any choice of ordered cover induces a hierarchy-based partition model $\hat{F}_{o,\lambda(o)}$. Finding the best model for a given data set therefore reduces to a search for the best ordered cover with respect to some model scoring criterion, such as penalized likelihood. We will construct the cover by means of three basic operations — partitioning, merging, and ghosting — which are closely related to the tree splitting and pruning operations in classification and regression trees (Breiman et al., 1984; Ripley, 1996).

The *partition* operation splits an existing partition $k$ into two by cutting out a subclass $s$, ie, it divides the partition $\pi_k$ into the disjoint subsets $\pi_k \cap s$ and $\pi_k - s$ where $s$ is some subclass of the partitioned class $c_k$. The new cover produced by applying the partitioning operation to the old cover $c_1^n = (c_1, \ldots, c_n)$ is given by the concatenation:

$$\text{part}_k(c_1^n, s) := c_1^{k-1}.s.c_k^n \qquad \text{(partitioning)}$$

ie, the partitioning results in the ordered cover $(c_1, \ldots, c_{k-1}, s, c_k, \ldots, c_n)$. We refer to the depth of $s$ relative to $c_k$ as the *depth* of the partitioning operation.

The *merge* operation deletes a partition from the cover and merges it into the subsequent partitions in the parent tree. The new cover produced by merging partition $k$ (with $k \neq n$) in the ordered cover $o_1^n$ is given by:

$$\text{merge}_k(c_1^n) := c_1^{k-1}.c_{k+1}^n \qquad \text{(merging)}$$

ie, the merging results in the ordered cover $(c_1, \ldots, c_{k-1}, c_{k+1}, \ldots, c_n)$, where $\text{par}_k$ becomes the new parent for the child partitions in $\text{chi}_k$.

The *ghost* operation can be viewed as a special instance of merging, in which the merged partition is replaced with the empty class $\perp$, but is retained in the tree structure as a ghost partition in order to make the computation of the partition index more efficient, as explained in section 3. The new cover produced by ghosting partition $k$ in the ordered cover $c_1^n$ is given by:

$$\text{ghost}_k(c_1^n) := c_1^{k-1}.\ \perp^{c_k} .c_{k+1}^n \qquad \text{(ghosting)}$$

where $\perp^c$ denotes the empty class annotated with ghost class $c$, ie, the ghosting results in the ordered cover $(c_1, \ldots, c_{k-1}, \perp^{c_k}, c_{k+1}, \ldots, c_n)$. It is easy to prove that any ordered tree cover can be produced from the trivial cover $(\top)$ by applying a sequence of partitioning, merging, and ghosting operations, even if the partitioning operations are restricted to depth 1. Figure 2 shown previously on page 4 illustrates how ordered covers can be constructed by means of sequences of partitionings, mergings, and ghostings.

```
Input: ordered tree cover $o$, partition node $k$
       in $o$, observations $\bar{x}_k$.
Output: ordered tree cover bestCover

bestCover := $o$;
bestScore := score of $o$;
repeat
    lastCover := bestCover;
    foreach $s$ among the biggest transitive
    subclasses of $c_k$ with respect to
    $\bar{x}_k$(lastCover) down to fixed depth do
        newCover := $\text{part}_k$(lastCover, $s$);
        newScore := score of newCover;
        if newCover is non-degenerate and
        newScore $>$ bestScore then
            bestScore := newScore;
            bestCover := newCover;
    if lastCover $\neq$ bestCover then
        $p$ := new partition in bestCover;
        bestCover :=
        FindCover(bestCover, $p$, $\bar{x}_p$(bestCover));
until bestCover = lastCover ;
if $|\bar{x}_k| <$ minimum partition size then
    bestCover := $\text{ghost}_k$(bestCover);
return bestCover;
```

**Algorithm 2**: FindCover($o$, $k$, $\bar{x}_k$)

Our algorithm for finding an ordered tree cover is listed in Algorithm 2. The search is performed in a greedy depth-first manner starting from the root partition in the trivial cover $(\top)$, and proceeds by recursively partitioning and ghosting partitions in the cover until we are unable to improve the score any further. The depth-first order is an advantage because it guarantees that we do not need to recalculate partition weights to the left of the current partition when new partitions are created. The algorithm expands a partition $k$ by computing the $m$ subclasses of $c_k$ with the largest data count down to a certain depth, and attempting a partitioning with each subclass. The subclass that results in the best score is then chosen for recursive partitioning, and the procedure is repeated until no more improvements can be found. The minimum partition size and the maximal partitioning depth are training parameters. The only limitation in the search is that we do not allow partitions to be degenerate (ie, to have lower partition weight than their parent partition), and that we ghost partitions whose data counts fall below the required minimum partition size.

To illustrate the FindCover algorithm, we now describe the actions taken by the algorithm in the construction of the ordered cover shown in Figure 2. Initially, we call the algorithm on the root partition in the trivial cover ($\top$). The algorithm examines a number of possible partitionings of $\top$, and chooses $c_1$ as the best one (step 1). The algorithm is applied recursively to $c_1$, which leads to a partitioning of $c_1$ with $c_{11}$ (step 2). The algorithm cannot find any improvements by partitioning $c_{11}$, and therefore moves back to $c_1$ and performs a second partitioning of $c_1$ with $c_{12}$ (step 3). Afterwards, it ghosts $c_1$ because it has become degenerate (step 4). Finally, the algorithm moves back to $\top$ and performs a second partitioning with $c_2$ (step 5).

For efficiency reasons, our current implementation of the FindCover algorithm does not immediately recalculate the interpolation mass for all existing child partitions whenever we split off a new subpartition, but only recalculates the interpolated mass and partition weights when we have arrived at a locally optimal cover. This means that the cover returned by FindCover may turn out to be degenerate, and for this reason, a call to FindCover is always followed by a sanitization operation which runs through the cover and ghosts or merges degenerate partitions. By reducing the number of partitions and gathering data points, the sanitization operation may open up the possibility for further partitioning. In our current implementation, we therefore rerun the FindCover algorithm after the sanitization operation, and repeat this cycle a small number of times. But the resulting improvements tend to be rather minor.

There are several possible choices of model scoring criterion. In the experiments, we have chosen to minimize the *Bayesian Information Criterion* (Schwarz, 1978) with an added penalty weight $\alpha$:

$$\mathrm{BIC}_\alpha(\hat{F}_{o,\lambda}|\bar{x}) := -2\ln L(\hat{F}_{o,\lambda}|\bar{x}) + \alpha|o|\ln|\bar{x}|$$

where $L(\hat{F}_{o,\lambda}|\bar{x}) := \prod_{x\in\bar{x}} \hat{F}_{o,\lambda}(x)$ is the likelihood function, $|\bar{x}|$ is the number of observations in $\bar{x}$, and $|o|$ is the number of free parameters in the model (one parameter for the cover size, and $|o|-1$ parameters for the partition weights). $\mathrm{BIC}_1$ corresponds to standard BIC, whereas $\mathrm{BIC}_0$ corresponds to maximum likelihood estimation.

## 6 Experiments

We now present the experiments we have performed in order to evaluate the performance of hierarchy-based partition models (HPMs) as density estimators. HPMs can be used to estimate joint and conditional densities for a wide range of estimation problems, including classification and reasonably low-dimensional categorical estimation problems. In this paper, we have chosen to focus on the estimation of bigram probabilities of the form $P(w_n|w_{n-1})$ where $w_{n-1}$ and $w_n$ are adjacent words in a text. Bigram estimation is interesting in terms of model evaluation because the sample space is large enough to lead to severe sparse data problems, even though the sample space is merely two-dimensional. Moreover, the problem is well understood with well-established state-of-the-art methods and easy access to large quantities of data. In the following, we describe the classification hierarchies, data, and evaluation measure we have used, and report the results of each of the four experiments.

### 6.1 Bigram data

As our data sets, we used the Europarl texts (Koehn, 2005) for English, Danish, and Finnish, three languages with varying degrees of morphological complexity. For each language, we constructed three bigram datasets by extracting all bigrams from the first 1,000, 10,000, and 100,000 lines of text in random order. For each bigram data set, we used the first 80% as our training set, and reserved 10% as a development set and 10% as an evaluation set. Table 1 shows the number of tokens and unique words in each of the data sets. The size of the data sets in our experiments was mainly limited by the time and memory requirements of our current HPM implementation.

### 6.2 Evaluation measure

The HPM models were evaluated by computing the perplexity of the unseen evaluation data, using modified Kneser-Ney smoothing (MKN) trained with SRILM (Stolcke, 2002) as our baseline. The perplexity of MKN is calculated by SRILM, whereas the perplexity of HPM is calculated by our own program. The program can read ARPA-format language model files, and we have verified that it returns the same perplexity scores as SRILM. We have also excluded the possibility of deficient HPM models by verifying that the prob-

|  | English |  |  | Danish |  |  | Finnish |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  | 1k | 10k | 100k | 1k | 10k | 100k | 1k | 10k | 100k |
| Training | 15 | 148 | 1,446 | 14 | 136 | 1,343 | 10 | 104 | 1,017 |
| Evaluation | 2 | 19 | 180 | 2 | 16 | 164 | 1 | 12 | 125 |
| Unique words | 4 | 19 | 65 | 5 | 23 | 100 | 5 | 34 | 172 |

Table 1: The number of bigrams in the training and evaluation sets and the number of unique words (in thousands).

ability assigned to $(\top, \top)$ was consistently equal to 1 in our experiments.

### 6.3 Implementation

Our implementation of Hierarchy-Based Partition Models was created in Java, and released under an open-source license (HPM, 2009). The experiments were performed on a Linux server with 4 CPUs and 8GB RAM, and lasted up to several days for the largest datasets. The current implementation has not been optimized with respect to processing time and memory use, and we have a number of ideas for further optimization. It is quite likely that these optimizations will lead to significant improvements in terms of both processing time, memory use, the size of the datasets, and the number of dimensions that can be handled by algorithm.

### 6.4 Classification hierarchies

We used a word suffix hierarchy as our hierarchy for unigrams, and a product word suffix hierarchy as our hierarchy for bigrams. Although we could have used Wordnet or any word class hierarchy instead, we chose to focus on word suffix hierarchies in our experiments because they do not contain any information which is not present in the text itself. This makes it easier to compare our method with state-of-the-art language models such as modified Kneser-Ney smoothing (Chen and Goodman, 1998).

### 6.5 Background distribution

We constructed two unigram estimators on the basis of the unigrams in the training data: for the predicted word, we used the empirical distribution of word suffixes; for the preceding word, we used an empirical distribution smoothed with $\frac{1}{n}$-add smoothing where $n$ was the number of unique words in the training set. As our background distribution, we used the product of the two unigram distributions, ie, our background distribution assumes the two words are independent. The choice of unigram distributions ensures that we get zero probabilities for exactly the same bigrams as the MKN baseline.[2]

### 6.6 Experiment 1: HPM vs Kneser-Ney

In order to determine how HPM compares with modified Kneser-Ney smoothing, we compared HPM with MKN on the three English, Danish, and Finnish bigram datasets. In all four experiments, we did not compute the optimal absolute discounts for HPM, but simply reused the absolute discounts computed by SRLIM for MKN. The other HPM model parameters were set to the values that produced the best results in other experiments, ie, minimum partition size 2 and BIC weight 0 (corresponding to maximum likelihood estimation). In Experiment 1, we restricted the partitioning search in the FindCover algorithm to the 20 largest subclasses down to depth 10. In Experiments 2-4, we used the 10 largest subclasses down to depth 10.

Tables 2–4 show the perplexities of the HPM and MKN models with respect to prediction (evaluation set) and goodness-of-fit (training set). The prediction data show that HPM outperforms MKN in all cases. Of the three languages, English has the simplest morphology, and Finnish the most complex, and the improvement in perplexity clearly correlates with the morphological complexity of the language. This is no surprise, since morphologically rich languages have a larger vocabulary than languages with little morphology, and the potential benefit of using word suffixes rather than whole words is therefore larger for morphologically rich languages. This observation probably also explains why the HPM improvement is slightly bigger on smaller data sets, where

[2]In Kneser-Ney smoothing, the lower-order Kneser-Ney model has been shown to result in better estimates of marginal probabilities than the empirical distribution (Goodman, 2002). It is quite conceivable that a similar result might hold for the choice of background distribution in Hierarchy-based Partition Models. However, we have not yet tried replacing our background distribution with a lower-order Kneser-Ney model instead of the empirical distribution.

| | English | | |
|---|---|---|---|
| | 1k | 10k | 100k |
| Prediction MKN | 132.47 | 164.29 | 145.61 |
| Prediction HPM | **123.15** | **155.50** | **143.66** |
| Goodness-of-fit MKN | 34.86 | 56.01 | 77.83 |
| Goodness-of-fit HPM | 107.08 | 119.18 | 118.27 |
| HPM cover size | 3,583 | 30,278 | 160,229 |

Table 2: Perplexity wrt. prediction and goodness-of-fit, and HPM cover size (Experiment 1, English).

| | Danish | | |
|---|---|---|---|
| | 1k | 10k | 100k |
| Prediction MKN | 153.17 | 191.87 | 189.49 |
| Prediction HPM | **130.54** | **167.20** | **174.42** |
| Goodness-of-fit MKN | 37.83 | 61.08 | 89.54 |
| Goodness-of-fit HPM | 124.28 | 152.85 | — |
| HPM cover size | 3,502 | 25,573 | 170,575 |

Table 3: Perplexity wrt. prediction and goodness-of-fit, and HPM cover size (Experiment 1, Danish).

| | Finnish | | |
|---|---|---|---|
| | 1k | 10k | 100k |
| Prediction MKN | 292.81 | 585.38 | 754.46 |
| Prediction HPM | **196.91** | **423.99** | **576.96** |
| Goodness-of-fit MKN | 45.95 | 59.92 | 104.97 |
| Goodness-of-fit HPM | 247.75 | 431.53 | — |
| HPM cover size | 2,787 | 25,600 | 217,072 |

Table 4: Perplexity wrt. prediction and goodness-of-fit, and HPM cover size (Experiment 1, Finnish).

the data are more sparse. Finally, we observe that MKN tends to provide a better fit to the training data than HPM, as evidenced by the goodness-of-fit data, which coupled with the better prediction in HPM models suggests that HPM models are less prone to overfitting than MKN.

## 6.7 Experiment 2: absolute discounting

In order to determine how absolute discounting affects HPM, we reran the HPM experiments without absolute discounting. Table 5 shows the perplexity scores for Danish with 1k and 10k lines of text.[3] Without absolute discounting, the HPM perplexity improvements relative to MKN disappear, which suggests that HPM tends to overfit when absolute discounting is turned off.

[3]For time reasons, we only conducted Experiments 2–4 for Danish (the middle language wrt. morphological complexity) with 1k and 10k lines of text. The complete set of results will be included in the final paper.

| | 1k | 10k |
|---|---|---|
| HPM –discounting | 149.82 | 188.58 |
| HPM +discounting | **132.67** | **167.56** |
| MKN | 153.17 | 191.87 |

Table 5: How absolute discounting affects HPM perplexity in prediction (Exp. 2 for Danish).

## 6.8 Experiment 3: minimum partition size

In order to determine how minimum partition size affects HPM, we repeated the HPM experiments with minimum partition sizes 1, 2, and 3. Table 6 shows the perplexity scores for Danish with 1k and 10k lines of text. The HPM estimator seems to underfit with minimum partition size 3, but overfits with minimum partition size 1, which makes a minimum partition size of 2 the optimal choice. The minimum partition size also affects the training time heavily, which is not surprising since a bigram with $n$ letters in one word and $m$ letters

|  | 1k | 10k |
|---|---|---|
| HPM mps=1 | 137.35 | 329.32 |
| HPM mps=2 | **132.68** | **167.56** |
| HPM mps=3 | 150.53 | 180.22 |
| MKN | 153.17 | 191.87 |

Table 6: How minimum partition size affects HPM perplexity in prediction (Exp. 3 for Danish).

|  | 1k | 10k |
|---|---|---|
| HPM bic=1 | 145.17 | 190.79 |
| HPM bic=0.5 | 143.16 | 187.02 |
| HPM bic=0.25 | 141.09 | 183.36 |
| HPM bic=0.1 | 138.35 | 178.68 |
| HPM bic=0 | **132.68** | **167.56** |
| MKN | 153.17 | 191.87 |

Table 7: How BIC penalty weights affect HPM perplexity in prediction (Exp. 4 for Danish).

in the other leads to $nm$ possible partitions when the partition size is 1. So for each bigram that occurs only once, a large number of partitions can be excluded from the search space when increasing the partition size from 1 to 2, although the search depth mitigates this problem to some extent.

### 6.9 Experiment 4: BIC penalty weights

In order to determine how the BIC penalty weights affect HPM, we repeated the HPM experiments with BIC penalties $0, 0.1, 0.25, 0.5, 1$ for HPM with absolute discounting. Table 7 shows the perplexity scores for Danish with 1k and 10k lines of text. The experiments show that BIC tends to underfit the data. Without absolute discounting (not shown), the optimal choice of BIC penalty is 0.5 and 0.25 for 1k and 10k lines of text, respectively, but the improvement over the maximum likelihood estimate is relatively small. This suggests that BIC penalizes free parameters too heavily, and that we should explore model selection criteria that place a smaller penalty on free parameters, such as the Akaike Information Criterion (Akaike, 1974) or modern versions of the Minimum Description Length Principle (Grünwald, 2007).

## 7 Discussion

For language modelling, the problem we have chosen for evaluating our estimator in this paper, the experiments show that HPM consistently outperfoms modified Kneser-Ney when applied to bigrams with text sizes up to one million words. The experiments also show that the proper choice of minimum partition size and absolute discounting parameters is quite important. This suggests that HPM performance could be improved by computing optimal HPM discounting parameters from held-out data, instead of reusing the modified Kneser-Ney parameters computed by SRILM.

## 8 Related work

HPMs are closely related to classification and regression trees (Breiman et al., 1984), probabilistic decision lists (Yarowsky, 2000; Goodman, 2002), and the tree-based estimation approach proposed by (Li and Abe, 1998). It differs from these approaches by being a correction density estimator, and by using a very general notion of classification hierarchies to identify regions of similar outcomes in the sample space. HPMs borrow important ideas from language modelling, such as absolute discounting (Ney et al., 1994). Moreover, since our specific implementation of HPMs for language modelling are based on suffix sequences rather than whole word sequences, and since HPM suffix hierarchies can be viewed as primitive clustering models, our language modelling HPMs resemble skipping models (Rosenfeld, 2000; Huang et al., 1993; Ney et al., 1994) and clustering models (Brown et al., 1992; Ney et al., 1994; Goodman, 2001) in important respects. Determining how ideas from these approaches can be used to improve HPM estimation in general, and how HPM estimation compares with these approaches when applied to language modelling and other problem areas in natural language processing and statistics, is an important topic for future research.

## 9 Conclusion

We have proposed hierarchy-based partition models as a new correction-based machine learning method for categorical data with associated classification hierarchies, and evaluated the new model in a series of experiments with bigram data sets. The experiments demonstrate that HPM consistently outperforms modified Kneser-Ney smoothing, ie, they show that there are statistical estimation problems for which HPM may be a useful alternative to other methods, and that classification hierarchies can provide useful information about the structure of a categorical sample space.

The main finding in this paper, in our view, is that existing state-of-the-art techniques in statistical natural language processing, or at least some areas of it, can be improved by exploiting the information that is hidden in the classification hierarchies that we construct routinely in linguistics and computational linguistics, but are unable to utilize fully with the statistical estimators that we normally use. In order to release this potential, we need a new set of statistical methods that can exploit classification hierarchies directly, rather than falling back on count-based methods or feature transformations to metric spaces that have no natural interpretation in the underlying categorical sample spaces. The methods presented in this paper are not a final solution to this problem, but rather one possible first step in this direction.

## 10 Future work

The idea of using classification hierarchies in statistical estimation is relatively unexplored in both statistics and natural language processing, and there are a lot of unsolved problems in our current account of Hierarchy-Based Partition Models. From a statistical point of view, the quality of the HPM estimates can probably be improved by incorporating ideas from related approaches, such as cost-complexity pruning in classification and regression trees (Breiman et al., 1984; Ripley, 1996) and the controlled marginals in Kneser-Ney smoothing (Chen and Goodman, 1998). For language modelling, the specific problem area we have focused on in this paper, there is a wide range of ideas from state-of-the-art language models that could probably be exploited in combination with HPM estimation, including ideas from caching models (Kuhn, 1988), sentence mixture models (Iyer and Ostendorf, 1996), model pruning (Stolcke, 1998; Seymore and Rosenfeld, 1996; Goodman and Gao, 2000), and the highly space-efficient use of Bloom filters (Talbot and Osborne, 2007). Some of these ideas might conceivably be useful outside language modelling as well. Finally, from a computational point of view, both training and estimation is painfully slow in our current implementation, which is the main reason why experiments with larger data sets have not been performed. It is quite likely that the speed of both training and estimation can be increased drastically, both by making a more efficient implementation, but also, and more importantly, by improving the way we search for an optimal cover.

Although language modelling is an interesting testbed for HPM estimation, it is quite likely that HPMs are even more useful in estimation problems characterized by low-dimensional categorical variables, large sample spaces, and smaller data sets than in language modelling — eg, in tasks where the data come from manually annotated sources such as treebanks, or in domain adaptation tasks where correction estimators can utilize their corrective nature. Exploring the usefulness of HPMs in these problem domains is another important area of future research.

## References

Akaike, Hirotugu. 1974. A new look at the statistical model identification. *IEEE Transactions on Automatic Control* 19(6):716–723.

Breiman, Leo, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. 1984. *Classification and regression trees*. Wadsworth.

Brown, Peter F., Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based $n$-gram models of natural language. *Computational Linguistics* 18(4):467–479.

Buch-Kromann, Matthias. 2006. Discontinuous Grammar. A dependency-based model of human parsing and language learning. Dr.ling.merc. dissertation, Copenhagen Business School.

Chen, Stanley and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Tech. Rep. TR-10-98, Harvard University.

Fellbaum, Christiane, ed. 1998. *Wordnet: an electronic lexical database*. Bradford Books.

Friedman, Jerome and Nicholas I. Fisher. 1999. Bump hunting in high dimensional data. *Statistics and Computing* 9:123–143.

Goodman, Joshua. 2001. A bit of progress in language modelling: extended version. Tech. Rep. MSR-TR-2001-72, Machine Learning and Applied Statistics Group, Microsoft Research.

Goodman, Joshua. 2002. An incremental decision list learner. In *Proc. EMNLP 2002*.

Goodman, Joshua and Jianfeng Gao. 2000. Language model size reduction by pruning and clustering. In *Proc. ICSLP 2000*, pages 110–113.

Grünwald, Peter D. 2007. *The Minimum Description Length principle*. MIT Press.

Halmos, Paul. 1974. *Measure theory*. Springer.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning. Data mining, inference, and prediction*. Springer, 2nd edn.

HPM. 2009. XHPM java implementation of HPM, rev. 326. http://open-source-dependency-toolkit.googlecode.com/svn/trunk/XHPM-EXP.

Huang, Xuedong, Fileno Alleva, Mei-Yuh Hwang, and Ronald Rosenfeld. 1993. An overview of the SPHINX-II speech recognition system. In *Proc. HLT 1993*, pages 81–86. ISBN 1-55860-324-7.

Iyer, R. and M. Ostendorf. 1996. Modeling long distance dependence in language: Topic mixtures vs. dynamic cache models. In *IEEE Transactions on Speech and Audio Processing*, pages 236–239.

Kneser, Reinhard and Hermann Ney. 1995. Improved backing-off for $m$-gram language modeling. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 1, pages 181–184.

Koehn, Philipp. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proc. MT Summit 2005*.

Kuhn, Roland. 1988. Speech recognition and the frequency of recently used words: a modified markov model for natural language. In *Proc. ACL 1988*, pages 348–350. Morristown, NJ, USA: Association for Computational Linguistics. ISBN 963 8431 56 3.

Li, Hang and Naoki Abe. 1998. Generalizing case frames using a thesaurus and the MDL principle. *Computational Linguistics* 24(2):217–244.

Ney, Hermann, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modeling. *Computer, Speech, and Language* 8:1–38.

Ripley, Brian D. 1996. *Pattern recognition and neural networks*. Cambridge University Press.

Rosenfeld, Ronald. 2000. Two decades of statistical language modeling: Where do we go from here. In *Proc. IEEE*, vol. 88, pages 1270–1278.

Schwarz, Gideon E. 1978. Estimating the dimension of a model. *Annals of Statistics* 6(2):461–464.

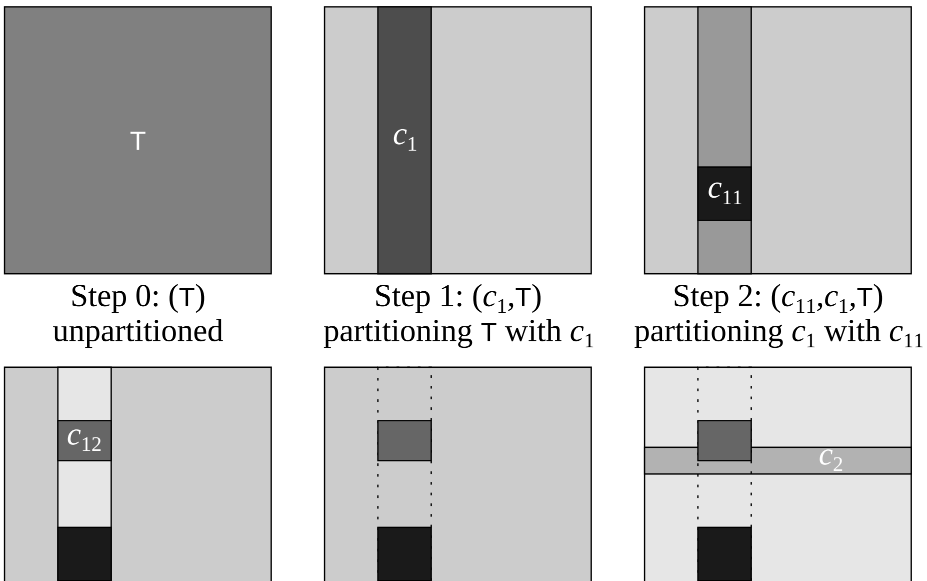
Seymore, Kristie and Ronald Rosenfeld. 1996. Scalable backoff language models. In *Proc. ICSLP 1996*, vol. 1, pages 232 – 235.

Stolcke, Andreas. 1998. Entropy-based pruning of backoff language models. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274.

Stolcke, Andreas. 2002. SRILM – an extensible language modeling toolkit. In *Proceedings of ICSLP*, vol. 2, pages 901–904. Denver, USA.

Talbot, David and Miles Osborne. 2007. Smoothed Bloom filter language models: Tera-scale LMs on the cheap. In *Proc. EMNLP 2007*.

Yarowsky, David. 2000. Hierarchical decision lists for word sense disambiguation. *Computers and the Humanities* 34(2):179–186.


⊤
$c_1$
$c_{11}$
Step 0: (⊤)
unpartitioned
Step 1: ($c_1$,⊤)
partitioning ⊤ with $c_1$
Step 2: ($c_{11}$,$c_1$,⊤)
partitioning $c_1$ with $c_{11}$
$c_{12}$
$c_2$