Modeling Text with Decision Forests using Categorical-Set Splits

Mathieu Guillame-Bert, Sebastian Bruch, Petr Mitrichev, Petr Mikheev, Jan Pfeifer*
Google

ABSTRACT

Decision forest algorithms model data by learning a binary tree structure recursively where every node splits the feature space into two regions, sending examples into the left or right branches. This "decision" is the result of the evaluation of a condition. For example, a node may split input data by applying a threshold to a numerical feature value. Such decisions are learned using (often greedy) algorithms that attempt to optimize a local loss function. Crucially, whether an algorithm exists to find and evaluate splits for a feature type (e.g., text) determines whether a decision forest algorithm can model that feature type at all. In this work, we set out to devise such an algorithm for textual features, thereby equipping decision forests with the ability to directly model text without the need for feature transformation. Our algorithm is efficient during training and the resulting splits are fast to evaluate with our extension of the QuickScorer inference algorithm. Experiments on benchmark text classification datasets demonstrate the utility and effectiveness of our proposal.

CCS CONCEPTS

• **Computing methodologies** → **Classification and regression trees**;

KEYWORDS

Decision Forests, Decision Tree Algorithm, Text Classification

1 INTRODUCTION

Machine Learning (ML) algorithms often consume observations in the form of feature vectors in order to train models. The semantics of each feature defines how it should be consumed by the algorithm and ultimately used in the resulting model. A typical decision tree algorithm, for example, learns to model data with a binary tree where each node recursively bifurcates the training examples according to a "split" of the feature space; how an optimal split is found for a feature depends entirely on semantics.

ML algorithms either have the ability to consume a feature type as is or otherwise require that it be transformed to a supported type. Virtually all algorithms, decision trees included [6, 16, 19], are able to ingest numerical features, sometimes exclusively so. The record is mixed for other feature types such as graphs, time series, text, or categorical features. Neural networks, for example, require that categorical features be transformed to one-hot vectors or another numerical form. On the other hand, decision forests that use a CART [6] split-finding algorithm support categorical features naturally [5, 6, 29].

Whether or not a feature transformation is used to prepare training data may dramatically affect model training. A contrived transformation step may remove potentially meaningful signals or, conversely, introduce meaningless correlations where none exists. One-hot encoded categorical features, for example, often lead to unbalanced splits in decision trees, leading to sub-optimal models that do not generalize as well as CART-powered decision trees—that is the driving reason for the use of CART in decision forest libraries such as LightGBM [20]. Such empirical observations have motivated researchers to study ways of extending decision trees to consume other feature types such as time series [11, 31] and timestamped symbol sequences [17] among others.

In this work, we set out to enable decision trees to consume another common feature type that remains unsupported to date: categorical sets. A categorical-set feature value is defined as a set of categorical terms. For example, consider a data point that is represented by the following 4 features: $\{f_1 = 5, f_2 = \text{"}cat\text{"}, f_3 = \{\text{"}blue\text{"}, \text{"}red\text{"}, \text{"}green\text{"}\}, f_4 = \emptyset\}$. In this example, f_1 is a numerical feature, f_2 is a categorical feature, and f_3 and f_4 are two categorical-set features. Note that, an empty categorical-set feature value is semantically different from a missing value.

A decision tree learning algorithm equipped with a split-finding algorithm that is specialized for categorical-set features may naturally consume text, as text can be trivially (though incompletely) expressed in that form. Such an extension of decision trees, in turn, allows the application to text corpora of an array of decision forest algorithms such as Random Forest (RF) [5], Multiple Additive Regression Trees (MART) [15], Dropout Multiple Additive Regression Trees (DART) [30], and Extremely Randomized Trees [16].

Our work formalizes categorical-set splits and offers a greedy algorithm to find them efficiently. Our formulation of a categorical-set split tests the presence of any one of a set of terms (called "mask") in the feature value set: When the intersection of the mask and feature value is nonempty, the split decision is in the affirmative. This mask itself is built incrementally using a stochastic, greedy process guided by the decision tree loss function: From a subsample of vocabulary terms, the term that minimizes the loss the most is added to the mask. This process is repeated until the loss cannot be further reduced, at which point the resulting mask is our split.

Our contributions can be summarized as follows:

- We define and formulate splits (conditions) on categorical-set features in the context of decision trees;
- We propose an efficient algorithm to learn such splits;
- We report an empirical comparison of our proposed algorithm with methods that require feature transformation;
- We present an analysis of the stability of the algorithm's hyper-parameters; and,
- We extend the QuickScorer [26] algorithm for efficient inference of models with categorical-set splits.

The remainder of this paper is organized as follows. We begin with a brief review of the literature in Section 2. We present our

*Correspondence to: Mathieu Guillame-Bert <gbm@google.com>

proposed algorithm in Section 3 and evaluate it in Section 4. That is followed by a detailed analysis in Section 5. Section 6 provides details on efficient model inference. Finally, we conclude this work in Section 7.

2 BACKGROUND AND RELATED WORK

A decision forest is a collection of decision trees. A decision tree itself is typically a binary tree that routes an example recursively until a leaf is reached. The decision at every intermediate node to take the left or right branch is, in most cases, made based on a condition on a single feature. We refer to this condition as a *split*. For example, a split for a numerical feature typically compares the value of that feature with a threshold—that threshold and the comparison operator are what define the split. When a decision tree is learned, the training algorithm finds the best split (e.g., threshold for a numerical feature) for each node greedily, selecting the split that optimizes a given scoring function (e.g., information gain, Gini index). For brevity, we refer to such split finding algorithms as "splitters."

The Machine Learning literature offers many splitters that are suitable for numerical features [6, 16, 19], categorical features [5, 6, 29], time series [11, 31], and timestamped symbol sequences [17]. However, to the best of our knowledge, no published work has addressed the challenge of finding splits on categorical-set features—defined in Section 1. However, multiple methods exist that may be utilized to transform categorical-set features into numerical or categorical values.

BagOfWords replaces a categorical-set feature with a histogram: the count of occurrences of each term in the vocabulary D. More precisely, a feature value $X \subset D$ is replaced with a set of numerical features $\{f_i\}_{i=1}^{|D|}$ with $f_i = |X \cap \{d_i\}|$ where $d_i \in D$ is a term in the vocabulary.

A **fixed pre-trained representation**, also known as fixed pre-trained embedding, projects every term individually [27, 28], or the set of terms as a whole [12], into a multi-dimensional dense vector space where terms that are "close" in the original representation—for some implicit or explicit definition of closeness—are also close in the target space. Such functions can be learned with a neural network using back-propagation [12] or other algorithms [28] capable of learning intermediate representations.

A number of recent publications have explored joint training of decision forests and neural networks [2, 8, 14, 21, 22, 24] as a way to harvest the power of deep learning [23] to consume text [12], images [18], graphs [32] and sets [34]. While not demonstrated, DeepSet [34] is another neural network-based transformation that may be used to incorporate categorical-set features in a decision forest. Note, however, that in this work we are interested in enabling decision trees to consume categorical-set features without transformation of any kind, including representation learning using neural networks.

3 FINDING CATEGORICAL-SET SPLITS

Decision forest learning algorithms such as RF, MART, or DART all rely on a splitter subroutine to find an optimal split for every node in the tree. During training, splitters are called frequently to search a large space of feature values given a large number of training examples in order to arrive at a sensible split that optimizes a split scoring function. The resulting split must also be efficient to compute as, during evaluation or inference, every intermediate node in the tree must quickly determine which branch an example should be routed towards under tight latency constraints. As such, splits and splitters play an outsize role in the efficiency of the training and inference procedures. It is therefore imperative that any proposed split and splitter be computationally cheap.

Algorithm 1 Greedy algorithm for finding categorical-set splits

Input: Collection of categorical-set features $\mathcal{X}$ and labels $\mathcal{Y}$. Sampling rate $p \in (0, 1]$. Vocabulary $D = \{d_j\}_{j=1}^{m}$.

Output: Split mask M.

$\hat{D} \leftarrow \{d \mid d \in D \text{ with probability } p\}$
$M \leftarrow \emptyset$ ▷ Initial empty mask
while true do
  $\hat{d} \leftarrow \arg\max_{d \in \hat{D}} \text{SCORE}(\mathcal{X}, \mathcal{Y}, M \cup d)$
  if $\text{SCORE}(\mathcal{X}, \mathcal{Y}, M \cup \hat{d}) \leq 0$ **then**
    break ▷ No further improvement possible
  end if
  $\hat{D} \leftarrow \hat{D} \setminus \hat{d}$
end while

Splitters also affect the generalizability of decision forest models: an overzealous splitter that produces a split that overfits the training data leads to poor generalization. This behavior can be controlled with regularization (e.g. loss term, examples or feature sub-sampling). Alternatively or additionally, one could afford a certain degree of stochasticity to a splitter to prevent overfitting. For example, presenting a splitter with only a subsample of training examples or of feature values is one way to introduce uncertainty. We incorporate a similar idea in our proposed algorithm.

Algorithm 1 presents our proposed splitter for categorical-set features. To understand the algorithm, let us define its output first: a categorical-set split. Like splits on numerical features, a categorical-set split consists of an operator and a "threshold." We choose intersection as the operator and a fixed subset of the vocabulary as the threshold, or more appropriately, mask. A split on a categorical-set feature is a test of whether its intersection with the mask is nonempty. The following definition formalizes this concept.

Definition 1. *Given a vocabulary of possible terms D and a categorical-set feature $X \subset D$, a Categorical-Set Split is the result of $X \cap M \neq \emptyset$, where $M \subset D$, a* mask*, is a fixed set of terms associated with the split.*

The objective is then to construct a mask $M \subset D$ such that the split formulated above optimizes a split scoring function on a given set of training examples—a list of n categorical-set feature values $\mathcal{X} = \{X_i\}_{i=1}^{n}$ with $X_i \subset D$, and their corresponding labels $\mathcal{Y} = \{y_i\}_{i=1}^{n}$. We note that any scoring function may be used but typical examples are information gain for classification with Random Forests, and mean squared error for Gradient Boosted Decision Trees.

Let us now describe how the algorithm constructs M. It first samples a random subset $\hat{D} \subset D$ with each term sampled independently with probability p, a hyperparameter that controls the stochasticity of the split. M is then initialized to an empty set and, in an iterative

Table 1: Dataset statistics. Average number of terms by example shows the average size of deduplicated categorical-sets.

Dataset	#Examples	#terms/examples
Stf. sentiment treebank (SST)	68.8k	9.8
Product review (CR)	8k	20.1
Movie review (MR)	22k	21.6
Subjectivity status (SUBJ)	20k	24.6
Opinion-polarity (MPQA)	22k	3.1

process, accumulates one vocabulary term at a time such that the split score is maximized. The algorithm stops when no additional vocabulary term improves the score.

4 EXPERIMENTAL EVALUATION

This section reports the empirical evaluation of our proposed splitter algorithm on RF and MART algorithms on 5 public text classification datasets. We begin with a description of these datasets, list the methods under evaluation, and finally present and discuss the results.

4.1 Datasets

We consider 5 binary classification datasets of the cleaned Sentiment Analysis dataset repository [10]. Table 1 shows the names and statistics of our datasets. We tokenize the text features by white space, thereby representing each piece of text as a set of unigrams. Once classifiers are trained, we measure the Area Under the Receiver Operating Characteristic Curves (AUC), averaged in a 5-fold cross-validation scheme.

4.2 Methods

We consider several learning algorithms including Neural Networks (NN), Linear classifier (Linear), Random Forests (RF) and Multiple Additive Regression Trees (MART)—the last two being decision forest algorithms. In order to evaluate our proposal we make two measurements. In one, we measure the effectiveness of our categorical-set splits for decision forests, applied directly to the datasets. The second trains a model using the learning algorithms above with text features transformed using one of the following functions:

- **TargetMean**, inspired by CatBoost [29], replaces a categorical feature by the conditional label distribution of its values, estimated on the training set. For example, in a binary classification setting, the categorical value "A" is replaced by the ratio of positive label among examples with value "A";
- **MaxHash**, inspired by MinHash [7] and Bloom filters [4], replaces a categorical-set feature by the maximum of a hash function applied to each term separately. The resulting value can be treated categorically or numerically. The process is applied multiple times using different, randomly selected hash seeds. In other words, a categorical-set feature value $X \subset D$ is thus converted to a set of categorical features $\{f_i\}_{i=1}^{k}$ with $f_i = \max_{x \in X} h(x, h_i)$, $h(\cdot)$ a hash function, and h_i a random seed;
- **BagOfWords** as described in section 2;
- **OneHot**, reduces BagOfWords to categorical features; and,
- **PreTrained** is a 128-dimension term based text embedding [3] trained on the English Google News 200B corpus. [1]

In the sections that follow, we adopt the following naming format: Method names begin with the learning algorithm (e.g., RF) followed by a sequence of pre-processing steps (if any) separated by the plus sign. For example, "RF MaxHash+ TargetMean" indicates that a Random Forest model is trained where the raw text features were transformed using MaxHash first, followed by TargetMean. CatCart indicates that a categorical feature is consumed with the CART splitter [6]. Finally, the proposed method is denoted by GreedyMask.

Many of the hyperparameters we used in this work are set to reasonable default values guided by previous publications [9, 20, 29], while a subset (e.g., vocabulary size, sampling rate) are determined by a small-scale validation and fixed across experiments. The following provides a summary:

- Tokenization: We keep the 5000 most frequent terms that appear at least 5 times. This is computed independently on the training partition of each cross-validation iteration.
- RF: We train 500 trees with a maximum depth of 32; the number of features randomly chosen to find a split in a node is the square root of the total number of features.
- MART: shrinkage is set to .1; maximum depth is 6 and number of trees is set to 500 with early stopping using 10% of the training dataset as validation; feature subsampling is disabled; and we use exact splitting for numerical features.
- NN: 3 layers with 32 units each; batch size is 32; train for a maximum of 20 epochs; early stopping using 10% of the training dataset as validation; finally, we use the AdaGrad [13] optimizer.
- Linear: 32 examples per batch; train for 20 epochs with the AdaGrad optimizer.
- GreedyMask: sampling rate of .2. We provide an analysis of the effect of this hyperparameter in Section 5.2.

4.3 Results

Table 2 shows the AUCs of the methods under consideration (in rows) on all datasets (in columns), averaged over 5-fold cross-validation trials. We also report the mean and median rank of each method in the same table.

The results in Table 2 show that our proposed algorithm when applied to Random Forests leads to considerable gains: The method comes first in terms of median rank, and is the best performing method on 3 of the 5 datasets. However, the method ranks poorly (13/20) on the MPQA dataset, falling far behind Linear PreTrained. We believe this unusual gap is an artifact of the dataset itself: Sentences in MPQA are very short, rarely exceeding a handful of terms—the average number of terms per example, as shown in Table 1, is a measly 3.1. With so few terms, it is easy for CatCart and GreedyMask to overfit. PreTrained, in contrast, is at an advantage as its representations are learned using another, larger dataset, making it less prone to overfitting.

As anticipated, the impact of spre-trained embeddings depends on the dataset. Pre-trained embeddings perform well on the MPQA dataset—the top four approaches use embeddings—whereas on other datasets the advantage is somewhat limited. Interestingly,

[1] Available at https://tfhub.dev/google/nnlm-en-dim128/1

Table 2: Five-fold cross-validation AUC of the various methods on the 5 binary classification datasets. The mean AUC is accompanied by the standard deviation and, in parentheses, the rank of the method for each dataset. Our methods are bold-faced.

Method	Median Rank	Avg Rank	SST	MR	CR	MPQA	SUBJ
RF$_{\text{GreedyMask}}$	1	3.8	**.9636±.0039 (1)**	**.842±.0458 (1)**	**.8723±.0448 (1)**	.8432±.0181 (13)	.9673±.0104 (3)
MART BagOfWords	3	4.6	.9561±.00556 (3)	.8351±.0491 (2)	.8559±.0439 (3)	.8384±.0157 (14)	**.9691±.0094 (1)**
MART$_{\text{GreedyMask}}$	3	5.2	.958±.00399 (2)	.8327±.0522 (3)	.8522±.0483 (4)	.8374±.0134 (15)	.9681±.0092 (2)
Linear PreTrained	6	6.4	.9187±.0122 (15)	.8213±.0157 (8)	.8652±.0186 (2)	**.9342±.0115 (1)**	.9643±.013 (6)
RF MaxHash+TargetMean	7	6.8	.9407±.00766 (9)	.8302±.014 (4)	.8439±.0447 (7)	.8887±.0104 (6)	.9633±.013 (8)
MART MaxHash+TargetMean	7	8	.9375±.00764 (11)	.8293±.0188 (6)	.8305±.0293 (11)	.8897±.00715 (5)	.964±.0114 (7)
RF PreTrained	9	9	.9482±.00763 (7)	.8023±.0298 (13)	.8423±.0324 (9)	.9278±.00954 (3)	.9458±.0202 (13)
NN PreTrained	9	9.4	.9129±.0113 (16)	.7992±.0286 (14)	.8466±.0235 (6)	.9324±.0188 (2)	.963±.0149 (9)
Linear MaxHash+TargetMean	9	9.8	.8991±.0102 (17)	.8294±.0179 (5)	.7713±.121 (14)	.8634±.0121 (9)	.9662±.0106 (4)
MART PreTrained	10	9.4	.938±.00842 (10)	.8056±.0313 (11)	.8299±.0532 (12)	.9259±.0164 (4)	.9568±.0148 (10)
Linear MaxHash+OneHot	10	11	.9448±.00652 (8)	.8062±.0205 (10)	.7113±.471 (15)	.8763±.0114 (7)	.7953±.626 (15)
NN BagOfWords	11	10	.9308±.00643 (14)	.8073±.0384 (9)	.8486±.0641 (5)	.8578±.0195 (11)	.9539±.0131 (11)
Linear BagOfWords	12	12	.937±.00577 (12)	.7907±.0338 (18)	.8437±.054 (8)	.8593±.0169 (10)	.9469±.0193 (12)
NN MaxHash+TargetMean	12	11	.8921±.0112 (18)	.8279±.0175 (7)	.8146±.0604 (13)	.8448±.0194 (12)	.9651±.0119 (5)
NN MaxHash+OneHot	13	13	.9365±.00815 (13)	.8042±.0284 (12)	.7064±.463 (16)	.8724±.0153 (8)	.7905±.63 (16)
RF$_{\text{CatCart}}$MaxHash	17	14.6	.9535±.00495 (4)	.7913±.0167 (17)	.696±.459 (17)	.8246±.0152 (16)	.7717±.687 (19)
MART$_{\text{CatCart}}$MaxHash	17.5	15.1	.9512±.00447 (5.5)	.7922±.0271 (15.5)	.6707±.433 (19.5)	.822±.0127 (17.5)	.7735±.694 (17.5)
MART MaxHash+OneHot	17.5	15.1	.9512±.00447 (5.5)	.7922±.0271 (15.5)	.6707±.433 (19.5)	.822±.0127 (17.5)	.7735±.694 (17.5)
RF BagOfWords	19	16.4	.8063±.0072 (19)	.7416±.0405 (19)	.8309±.0268 (10)	.7365±.042 (20)	.9119±.0272 (14)
RF MaxHash+OneHot	20	19.4	.7794±.0219 (20)	.7166±.0226 (20)	.6864±.394 (18)	.7453±.0408 (19)	.7525±.517 (20)

linear models appear to yield higher AUCs when trained on pre-trained embeddings.

GreedyMask performs better with RF than with MART. This can be partially explained by the stochasticity in Random Forests: On datasets with a low example-to-feature ratio, RFs have low variance as individual decision trees only model a subsample of features. GreedyMask creates a large feature space, leading to effects similar to presenting the algorithms with a large number of features.

We note that, the ranking above does not take into account the differences in AUC between methods: Small differences matter as much as large ones. A more appropriate comparison would be to measure the gain towards the optimal AUC of 1 relative to a fixed baseline: $\frac{\text{accuracy}-\text{baseline}}{1-\text{baseline}}$. We call this quantity the *relative accuracy headroom reduction* (RAHR) between methods and use RF BagOfWords as baseline. This statistic also aids in the visualization of the results of our experiments by highlighting relative gains, as Figure 1 illustrates.

Our proposed GreedyMask with RF has an RAHR of .507, followed by Linear PreTrained (RAHR=0.482) and RF MaxHash+TargetMean (RAHR=.475). The overall order of RAHR and ranks are similar, with the main difference being between the top contenders: Linear PreTrained and RF MaxHash+TargetMean go from global ranks 4 and 5, respectively, to ranks 2 and 3 largely due to the large relative gain of these methods on the MPQA dataset.

5 ANALYSIS

In this section, we take a closer look at the methods considered in this work. We begin with a comparison of the structure of the learned models. We then examine the effect of hyperparameters on model performance.

5.1 Structure

Table 3 reports model statistics resulting from the utilization of different pre-processing transformations with the RF algorithm on two datasets. We note that similar conclusions can be drawn from the other 3 datasets, which we have omitted for brevity.

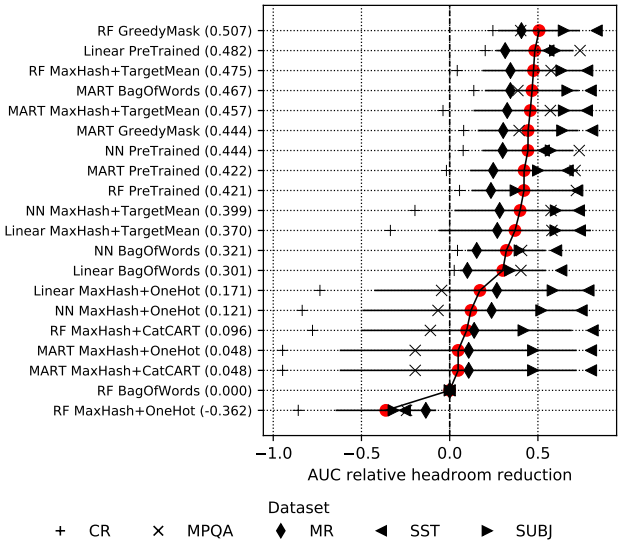

Figure 1: Relative accuracy headroom reduction (RAHR) of methods on 5 datasets relative to RF BagOfWords. The red circles traced across methods are mean RAHR of each method and the solid horizontal line around each circle represents one standard deviation in both directions.

Table 3: Structure statistics of the RF models for the MR and SST datasets. Balance ratio is the ratio of the log of the number of nodes per tree ($\log_2(\#Nodes/Tree)$) to the average depth—a balance ratio of 1 indicates a fully balanced tree.

Method	Avg Depth		#Nodes/Tree		Balance	
	MR	SST	MR	SST	MR	SST
RF$_{\text{GreedyMask}}$	11.5	17.5	477	3499	.771	.671
RF BagOfWords	20.5	21.6	815	1642	.472	.494
RF$_{\text{CatCart}}$MaxHash	12.0	12.6	369	855	.713	.775
RF MaxHash+TargetMean	13.5	17.0	1493	6109	.782	.739
RF MaxHash+OneHot	19.9	20.1	722	1099	.477	.503

We observe that BagOfWords and MaxHash+OneHot lead to deeper trees, while other solutions learn much shallower trees. One possible interpretation is that node splits resulting from features transformed using BagOfWords and MaxHash+OneHot afford little separability powers, and as a consequence, more splits are required to obtain better decision boundaries. It is also worth noting that BagOfWords effectively tests one term at a time, leading to larger trees that generalize poorly.

It does not come as a surprise then that BagOfWords and MaxHash+OneHot have significantly smaller balance ratios relative to other methods, indicating that trees are on average less balanced. This phenomenon too can be explained by the fact that splits consider a single term (or a single random hash) at a time, thereby repeatedly forcing training examples down the negative branch, ultimately resulting in unbalanced trees.

5.2 Hyperparameter Stability

Our proposed method has a single hyperparameter, a sampling rate p, which introduces randomness in the splitter. By incorporating this hyperparameter, we hoped to allow a form of regularization and prevent overfitting. In this section, we study the effect of the sampling rate on the final model across different datasets.

Figure 2 shows the change in mean AUC for different values of sampling rate. As before, AUCs are estimated with 5-fold cross-validation. Other hyperparameters are left unchanged (see Section 4), with the exception of the vocabulary size which is adjusted from 5000 to 2000 terms to facilitate faster experiments.

Model performance is relatively stable and does not change dramatically with changes in the sampling rate: Excluding the sampling rate of .01, the average difference between the best and worst AUCs for $p \in [0.05, 0.5]$ is only .0078. The optimal sampling rate naturally depends on the dataset, ranging from the smallest to the largest tested values. While not reported in Figure 2, for some datasets the optimal sampling rate appears to be 1.0, meaning no sampling at all. On average, however, $p = .2$ is a reasonable default value for RF with an average .0026 AUC drop from the best setting.

Confirming the results of Section 4, RF performs better than MART by an average 0.0063 in AUC. By construction, RF is less prone to overfitting than MART and, as such, can better correct our splitting algorithm's tendency to overfit to the training data. More work, however, is required to understand and improve our proposed solution for use with MART.

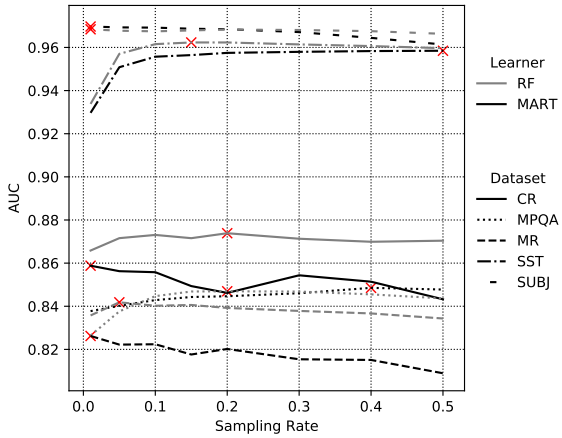

Figure 2: Mean AUC with respect to sampling rate. The cross represents the highest AUC for each method-dataset pair.

6 EFFICIENT MODEL INFERENCE

The naïve way to evaluate a decision tree on an input example is to start at the root and evaluate its condition in order to route the example to one of its (left or right) branches, and repeat that operation until we reach a leaf [1]. Though trivial to implement, that approach is suboptimal. Researchers have thus developed more optimized inference algorithms such as QuickScorer [26] (QS) and its extensions V-QuickScorer [25] (v-QS) and RapidScorer [33]. The core idea is to evaluate a tree by evaluating all its nodes simultaneously, and then retrieving the active leaf. These methods, despite having an exponentially higher time complexity, run orders of magnitude faster than the top-down approach on modern CPUs because of their more predictable memory access pattern and branching.

In this section, we present an extension of QS to support conditions generated by Algorithm 1. To understand this extension, let us briefly describe the mechanism by which QS determines which leaf is active given an input example with numerical features: QS begins by constructing a "leaf mask," a bit vector, initially all set, whose size is equal to the number of leaves in the tree. Each node too has a "node mask," a bit vector of the same size that encodes the leaves that are unreachable if an example fails to satisfy that node's split condition. QS proceeds by taking one numerical feature at a time and iterating over all nodes that split on that feature. If the node's threshold is smaller than the feature value, that node's mask is applied (with AND) to unset unattainable leaves. In the end, the index of the lowest set bit in the leaf mask is the active leaf. Note that, a separate leaf mask is maintained per tree.

Algorithm 2 presents our extension of QS. We adopt the same notation as in Algorithm 1 in the original work [26] and refer the reader to that work for a complete account. For our algorithm to work, we prepare the following data structure for each categorical-set feature separately: We compile what we refer to as "term masks," bit vectors that are similar to node masks in QS but that encode, for each term in the vocabulary, the leaves that are unreachable if an example contains that term. Figure 3 shows an example decision forest with categorical-set splits along with its term masks.

Algorithm 2 Extension of QuickScorer for categorical-set splits. Notation and surrounding code follow Algorithm 1 in [26].

Input: An input example x.
Init leaf mask, `leafidx`, indexed by trees ▷ Lines 1–3 of [26]
Apply numerical node masks to leaf masks ▷ Lines 4–9 of [26]
for each Categorical-Set Feature f **do**
  for each $v \in x[f]$ **do** ▷ Terms present in feature f of x
    $r \leftarrow$ `termMaskIndex`$[f][v]$
    for each $w \in [r.begin, r.end)$ **do**
      $m \leftarrow$ `termMask`$[f][w]$
      `leafidx`$[m.treeId] \wedge = m.mask$
    end for
  end for
end for
Retrieve leaf indices ▷ Lines 10–15 of [26]

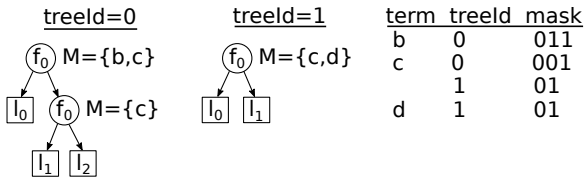

Figure 3: Illustration of `termMask` for an example forest with a single categorical-set feature f_0 with vocabulary $\{a, b, c, d\}$. If an example feature contains the term *c*, leaves l_0 and l_1 are not reachable in the first tree, resulting in the term mask 001**. Similarly, leaf l_0 from the second tree is not reachable, leading to the mask** 01**. The presence of *a* is inconsequential for the leaf mask of both trees. So is the presence of *b* for the second tree's leaf mask and *d* for the first tree's.**

Once term masks are built for all categorical-set features and all trees in the forest, we group them by feature and sort each group by term into an array. This is the `termMask` structure in Algorithm 2. Note that, in each group, a term may have more than one term mask as it may appear in splits in more than one decision tree. Finally, for each group, we index every term by storing its start and end indices in `termMask`. This organization results in a compact and access-efficient representation.

We put the extended QS of Algorithm 2 to the test. Table 4 reports its inference speed on the first fold of experiments in Section 4, and compares it with the direct top-down approach, VPred[1]. This benchmark considers only the application of a trained model on already-processed input features; in other words, we exclude the time required to perform tokenization or compute hashes or label statistics. The experiments are run with a single thread on a 3.70GHz Intel Xeon CPU. For VPred, both the model mask and the examples are sorted prior to the benchmark in order to evaluate the intersection condition in linear time with the number of items. The results are averaged over 100 runs over the entire dataset, and are preceded by 10 warm-up runs. Numerical splits are evaluated with SIMD instructions (v-QS). The categorical-set split evaluation does not rely on SIMD instructions.

Table 4: Inference speed per examples of the different MART models on the proposed v-QS and VPred implementation.

Method	Extended v-QS (Âţs)	VPred (Âţs)
MART$_{\text{GreedyMask}}$	.636	8.65
MART BagOfWords	.754	15.4
MART$_{\text{CatCart}}$MaxHash	2.80	2.16
MART MaxHash+TargetMean	1.59	4.65
MART MaxHash+OneHot	2.85	2.18

The QS implementation for categorical-set split is nearly 13x faster than the VPred implementation, demonstrating that such splits are well-suited for the QS algorithm. GreedyMask (withour SIMD instructions) runs nearly 20% faster than BagOfWords (with SIMD instructions).

7 CONCLUSION

In this work, we proposed a novel algorithm that enables decision forests to consume categorical-set features, effectively allowing them to model text without a need for feature transformation. Our solution equipped decision forests with the ability to efficiently find (greedy) splits in the space of sets of objects. We also extended QuickScorer, an inference algorithm to evaluate decision forests efficiently on modern CPUs, to include our proposed split.

Experiments on text classification showed that our method is competitive in terms of quality and inference speed compared to existing methods. Furthermore, an examination of the resulting models in terms of structure and sensitivity to our method's hyper-parameter shows that our proposed method yields balanced trees and its performance is stable across various datasets and settings.

This work gives rise to a number of future research directions. Having established the feasibility of consuming raw textual features with decision forests, we are interested in variants of the proposed algorithm (e.g., ngram-based splits) and in better understanding their effect on different decision forest algorithms (RF vs. MART). Preventing the splitter from overfitting to training data, particularly on small datasets, is a topic worth exploring. Another question left unanswered is the interpretability of our proposed split: How one systematically assesses the role a particular term or split plays in the model needs to be studied.

8 ACKNOWLEDGEMENTS

We extend our thanks to Vytenis Sakenas, Dmitry Osmakov, Alexander Grushetsky, and the members of the RankLab team at Google for helpful discussions and insight. We also thank Masrour Zoghi for reviewing an early draft of this work.

REFERENCES

[1] N. Asadi, J. Lin, and A. P. de Vries. 2014. Runtime Optimizations for Tree-Based Machine Learning Models. *IEEE Transactions on Knowledge and Data Engineering* 26, 9 (2014), 2281–2292.

[2] Randall Balestriero. 2017. Neural Decision Trees. *ArXiv* abs/1702.07360 (2017).

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* 3, null (March 2003), 1137âĂŞ1155.

[4] Burton H. Bloom. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (July 1970), 422âĂŞ426.

[5] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5âĂŞ32. https://doi.org/10.1023/A:1010933404324

[6] Leo Breiman, Joseph H Friedman, R. A. Olshen, and C. J. Stone. 1983. Classification and Regression Trees.

[7] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 2000. Min-Wise Independent Permutations. *J. Comput. System Sci.* 60, 3 (2000), 630 – 659.

[8] Sebastian Bruch, Jan Pfeifer, and Mathieu Guillame-bert. 2020. Learning Representations for Axis-Aligned Decision Forests through Input Perturbation. (2020). arXiv:cs.LG/2007.14761

[9] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785âĂŞ794.

[10] Alexis Conneau and Douwe Kiela. 2018. SentEval: An Evaluation Toolkit for Universal Sentence Representations. *arXiv preprint arXiv:1803.05449* (2018).

[11] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. 2013. A time series forest for classification and feature extraction. *Information Sciences* 239 (2013), 142 – 153. https://doi.org/10.1016/j.ins.2013.02.030

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[13] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (July 2011), 2121–2159.

[14] Ji Feng, Yang Yu, and Zhi-Hua Zhou. 2018. Multi-Layered Gradient Boosting Decision Trees. In *NeurIPS*.

[15] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Ann. Statist.* 29, 5 (10 2001), 1189–1232. https://doi.org/10.1214/aos/1013203451

[16] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely Randomized Trees. *Mach. Learn.* 63, 1 (April 2006), 3âĂŞ42. https://doi.org/10.1007/s10994-006-6226-1

[17] Mathieu Guillame-Bert and Artur Dubrawski. 2017. Classification of Time Sequences using Graphs of Temporal Constraints. *Journal of Machine Learning Research* 18, 121 (2017), 1–34. http://jmlr.org/papers/v18/15-403.html

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. 770–778. https://doi.org/10.1109/CVPR.2016.90

[19] Ruoming Jin and Gagan Agrawal. 2003. Communication and Memory Efficient Parallel Decision Tree Construction. In *In Proceedings of Third SIAM Conference on Data Mining*.

[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*. 3146–3154.

[21] Guolin Ke, Zhenhui Xu, Jia Zhang, Jiang Bian, and Tie-Yan Liu. 2019. DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 384âĂŞ394.

[22] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò. 2016. Deep Neural Decision Forests. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAIâĂŹ16)*. AAAI Press, 4190âĂŞ4194.

[23] Yann LeCun, Y. Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521 (05 2015), 436–44. https://doi.org/10.1038/nature14539

[24] Pan Li, Zhen Qin, Xuanhui Wang, and Donald Metzler. 2019. Combining Decision Trees and Neural Networks for Learning-to-Rank in Personal Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2032âĂŞ2040.

[25] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2016. Exploiting CPU SIMD Extensions to Speed-up Document Scoring with Tree Ensembles. 833–836. https://doi.org/10.1145/2911451.2914758

[26] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2017. *QuickScorer: Efficient Traversal of Large Ensembles of Decision Trees.* 383–387. https://doi.org/10.1007/978-3-319-71273-4_36

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, G.s Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems* 26 (10 2013).

[28] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543. https://doi.org/10.3115/v1/D14-1162

[29] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 6638–6648. http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf

[30] K. Rashmi and Ran Gilad-Bachrach. 2015. DART: Dropouts meet Multiple Additive Regression Trees. (05 2015).

[31] Juan RodrÃŋguez and Carlos Alonso. 2004. Interval and dynamic time warping-based decision trees. 548–552. https://doi.org/10.1145/967900.968015

[32] Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local Neural Networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), 7794–7803.

[33] Ting Ye, Hucheng Zhou, Will Zou, Bin Gao, and Ruofei Zhang. 2018. RapidScorer: Fast Tree Ensemble Evaluation by Maximizing Compactness in Data Level Parallelization. 941–950. https://doi.org/10.1145/3219819.3219857

[34] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. *ArXiv* abs/1703.06114 (2017).