ExploreKit: Automatic Feature Generation and Selection

Gilad Katz
University of California, Berkeley
giladk@berkeley.edu

Eui Chul Richard Shin
University of California, Berkeley
ricshin@berkeley.edu

Dawn Song
University of California, Berkeley
dawnsong@cs.berkeley.edu

***Abstract*—Feature generation is one of the challenging aspects of machine learning. We present ExploreKit, a framework for automated feature generation. ExploreKit generates a large set of candidate features by combining information in the original features, with the aim of maximizing predictive performance according to user-selected criteria. To overcome the exponential growth of the feature space, ExploreKit uses a novel machine learning-based feature selection approach to predict the usefulness of new candidate features. This approach enables efficient identification of the new features and produces superior results compared to existing feature selection solutions. We demonstrate the effectiveness and robustness of our approach by conducting an extensive evaluation on 25 datasets and 3 different classification algorithms. We show that ExploreKit can achieve classification-error reduction of 20% overall. Our code is available at *https://github.com/giladkatz/ExploreKit*.**

I. Introduction

Effective feature engineering serves as a prerequisite to many machine learning tasks. Success with learning algorithms requires the creation of features that provide useful insights into different aspects of the data while taking the idiosyncracies and limitations of the algorithms into account.

We present ExploreKit, a framework designed to alleviate difficulties involved in feature engineering through automation. Based on the intuition that highly informative features often result from manipulations of elementary ones, we identified common *operators* to transform each feature individually or combine several of them together. ExploreKit uses these operators to generate many *candidate features*, and chooses the subset to add based on the empirical performance of models trained with candidate features added.

Since evaluating all candidate features isn't feasible, we propose a novel machine learning approach to predict candidates' usefulness. Our approach models the interactions between the candidate features and the dataset as well as various aspects of the dataset itself. To the best of our knowledge this is the first attempt to address this problem using machine learning.

Our contributions are as following:

- We develop a modular framework for generating new candidate features through structured operations and evaluating them automatically.
- We present novel methods for efficiently evaluating the large set of generated candidate features, based on a machine learning approach that predicts the utility of any given feature.

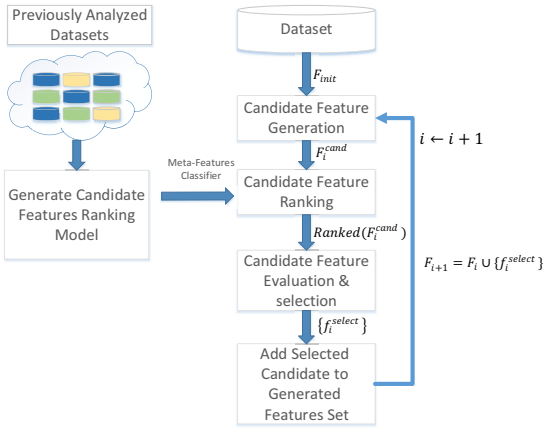

Fig. 1: ExploreKit system architecture

- We empirically demonstrate the merits of our approach on a large group of datasets and multiple classifiers, achieving approximately 20% error reduction.

II. Related Work

Feature Generation Using Deep Learning. In recent years, deep learning has outperformed other forms of machine learning in a variety of challenging problems such as image classification [8] and speech recognition [6]. Its success originates in its ability to operate directly on the raw data and learn a higher-level representation automatically. However, the construction of these representations by deep learning models is notoriously opaque to human analysts. In contrast, our approach creates a small set of features using human-understandable operations.

Automatic Feature Generation and Selection. To the best of our knowledge only one previous work outside the field of deep learning has attempted to automatically generate features to improve the performance of machine learning algorithms. The Data Science Machine [7] has the similar goal of generating new features to improve classification performance. Unlike this work, ours works on "flat" data without having to define entity relations. In addition, we use machine learning to predict the performance of candidate features, leveraging experience from previous datasets. Finally, our approach finds a small set of new features to reach the desired performance, while this work requires the generation of thousands.

2374-8486/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDM.2016.176

Algorithm 1 The ExploreKit Algorithm

procedure GENERATEFEATURES(dataset, maxIterations)
  $F_0 \leftarrow dataset$.**GetFeaturesSet**()
  for $i = 0$ to maxIterations $- 1$ **do**
    $F_i^{cand} \leftarrow$ **GenerateCandidateFeatures**(F_i); $RankedF_i^{cand} \leftarrow$ **RankCandidateFeatures**(F_i^{cand})
    $bestFeatureSoFar \leftarrow \emptyset$; $bestImprovementSoFar \leftarrow -\infty$; $F_{i+1} \leftarrow \emptyset$
    for each (candidateFeature, featureScore) in **SortDescending**$(RankedF_i^{cand})$ **do**
      if $featureScore < threshold_f$ **then**
        break
      $improvement \leftarrow$ **EvaluateWithClassifier**$(F_i) -$ **EvaluateWithClassifier**$(F_i \cup \{candidateFeature\})$
      if $improvement \geq \epsilon_w$ **then**
        $F_{i+1} \leftarrow F_i \cup \{candidateFeature\}$; ***break***
      if $improvement > bestImprovementSoFar$ **then**
        $bestImprovementSoFar \leftarrow improvement$; $bestFeatureSoFar \leftarrow candidateFeature$
    if $F_{i+1} = \emptyset$ **and** $bestImprovementSoFar \geq threshold_w$ **then**
      $F_{i+1} \leftarrow F_i \cup \{bestFeatureSoFar\}$
    else
      return F_i
  return $F_{\text{maxIterations}}$

III. Problem Definition

For our task, we assume a dataset of input-output pairs, where each instance comes with a set of original features. Our goal is to generate new features based on the original features, so that the learning procedure is able to find a function within its restricted class of functions that better approximates the true underlying input-output function.

More formally, we have:

- original features $F_{init} = \{f_1, \cdots, f_n\}$. For notational uniformity, we consider the output to be one of the features.
- Example instances I_{all} which we can split into I_{train}, $I_{holdout}$, and I_{test}.
- Dataset $D := I_{all} \times F_{init}$; for each instance, we have values for all of the original features.
- A class of functions $\mathcal{C}_F \subset \{I \times F \rightarrow Y\}$ considered by the learning procedure, where F is a set of features.
- A learning procedure $\mathcal{L}$, which takes $I \times F$ and produces a member of $\mathcal{C}_F$: $\mathcal{L}(I \times F) \in \mathcal{C}_F$.
- An evaluation procedure $\mathcal{E}(C_F, I \times F)$, which computes the error of a learned function C_F on $I \times F$.
- Potential new features $f_1^{cand}, f_2^{cand}, \cdots \in F_{cand}$.

Then we want to obtain

$$\underset{F_{select} \subset F_{cand}}{\arg\min} \quad \mathcal{E}(\mathcal{L}(I_{train} \times F_{init} \cup F_{select}), I_{holdout} \times F_{init} \cup F_{select}) \tag{1}$$

where F_{select} is the set of features selected from all potential features.

IV. The Proposed Method

Overview. Our feature generation process is presented in Figure 1 and Algorithm 1. It is an iterative process where each iteration comprises of three phases: *candidate features generation*, *candidate feature ranking*, and *candidate features evaluation & selection*.

In the candidate feature generation phase we apply multiple operators on the current set of features F_i to generate a large set of candidate features F_i^{cand}. In the candidate features ranking phase we assign a score to each candidate feature $f_{i,j}^{cand} \in F_i^{cand}$ based on its estimated contribution and produce an ordered list of features $RankedF_i^{cand}$.

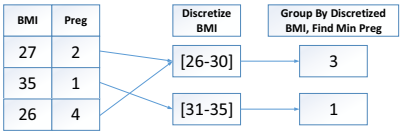

Fig. 2: An illustrations of the operators used by ExploreKit in the generation of a feature in the Pima dataset

As the number of candidate features is very large, we must use an efficient ranking function. Existing methods which do not require the training of a classification model, such as Information Gain, proved inadequate for the large feature space. Moreover, we hypothesize that effective ranking requires taking into account the size and feature composition of the analyzed dataset. We therefore propose a novel ML-based approach for candidate feature ranking. We define meta-features to represent both the dataset and the candidate feature and train a feature ranking classifier. This is, to the best of our knowledge, the first such attempt.

In the candidate features evaluation & selection phase we use greedy search to evaluate the ranked candidate features. We evaluate the performance of the joint set $F_i \cup \{f_{i,j}^{cand}\}$ for each $f_{i,j}^{cand} \in RankedF_i^{cand}$ and compute the reduction in classification error compared with F_i. When the performance improvement exceeds a predefined threshold ϵ_w, the evaluation process terminates and we select the current candidate feature, denoted as f_i^{select}. We define the joint set $F_i \cup \{f_i^{select}\}$ as the current feature set of the following iteration F_{i+1}. Next we describe the phases of this process in detail.

A. Generation of Candidate Features

The goal of this phase is to generate a large set of candidate features F_i^{cand} using the current features set F_i. We first present the operators used in the candidate feature generation and then describe our proposed process.

1) ***Operator Types****:* We apply three types of *operators* to generate the candidate features set F_i^{cand} for iteration i: *unary*, *binary* and *higher-order*.

Unary operators: applied on a single feature. Each operator in this group belongs to one of two sub-groups:

- **Discretizers**: used to convert continuous and datetime features into discrete (i.e. categorical) ones. Discretization [4] is necessary in many popular classification algorithms (e.g., Decision Trees and Naive Bayes) and has also been shown to contribute to performance [1]. For our evaluation, we have implemented the *EqualRange* discretization for numeric features (partition the range of values of the feature into X equal segments) and the *DayOfWeek*, *MonthOfYear* and *IsWeekend* for date-time features.
  Another important benefit of discretization is that it provides us with transformations of continuous features that can be utilized by the higher–order operators.
- **Normalizers**: used to fit the scale of continuous (i.e. numeric) features to specific distributions. Normalization has also been shown critical to the performance of multiple machine learning algorithms [3]. For our experiments we have implemented normalization to the range [0,1].

Binary operators: applied on a pair of features. This group currently consists of the four basic arithmetic operations: $+, -, \times, \div$.

Higher-order operators: use multiple (two or more) features for the generation of a new one. We have implemented five operators in this group: *GroupByThenMax*, *GroupByThenMin*, *GroupByThenAvg*, *GroupByThenStdev* and *GroupByThenCount*. These operators implement the SQL-based operations with the same name.

It is important to point out that in our experiments we only use a small set of operators. Additional operators can be easily created and added to our framework, including many domain-specific operators for fields such as biology or time-series analysis.

*2) **Generating the Candidate Features Set**:* We generate the candidate features by applying the operators in the following order:

1. We apply the *unary operators* on all possible features in the features set. We use this step to create $F_{u,i}$, the normalized and discretized versions of all non-discrete features in F_i.
2. W apply the *binary* and *higher-order* operators on the unified set $F_i \cup F_{u,i}$. All possible valid feature combinations are generated. We denote the features generated in this step as $F_{o,i}$.
3. For every applicable (i.e. non-discrete) feature in $F_{o,i}$, we once again apply all the unary operators. We denote the features set generated by this step as $F_{ou,i}$.
4. The final candidate features set for iteration i is the union of all generated feature sets: $F_i^{cand} = F_{u,i} \cup F_{o,i} \cup F_{ou,i}$.

In order to limit the size of F_i^{cand}, the features are only combined once: none of the generated candidate features is re-used to generate additional candidate features.

Example: applying multiple operators. Consider the following example from the Pima dataset.[1] The goal is to identify women with risk of diabetes. Two features from this dataset are BMI (body mass index) and the number of pregnancies. One possible way to generate additional features is as follows:

1. Discretize (partition) the BMI attributes into segments: $[20, 25)$, $[25, 30)$ and so forth.
2. Group all instances based on their BMI assignment.
3. For the instances of each group, find the minimal number of pregnancies of any member of the group.

This feature may enable a classifier to infer a woman's relative number of pregnancies while using a smaller, more relevant reference group. This feature is created by using an unary discretization operator, followed by a *GroupByThenMin* operator. This process is illustrated in Figure 2.

B. Ranking Candidate Features

The goal of this phase is to use computationally-efficient ("lightweight") methods to rank the large number of candidate features F_i^{cand}. We develop a novel machine learning based approach that generates a set of *meta-features* $F_{i,j}^{meta}$ for each candidate feature $f_{i,j}^{cand}$. Once the meta-features are generated, we use the ranking classifier C to assign a score to each $f_{i,j}^{cand}$. We train C on previously analyzed datasets (the training process is described later in this section).

We generate two types of meta-features: *dataset-based* and *candidate features-based*.

Dataset-based meta-features. Every dataset has multiple characteristics that may affect the likelihood of $f_{i,j}^{cand}$ being effective. We generate four types of meta-features:

1. **General information**: general statistics on the analyzed dataset: number of instances and classes, statistics on the size and other statistics on F_i.
2. **Initial evaluation**: statistics on the current performance of the classifier when applied on F_i. The generated meta-features include AUC, log loss and precision/recall values at various thresholds.
3. **Entropy-based measures**: we partition F_i into subgroups based on their type (discrete, numeric, date-time) and calculate statistics on the Information Gain (IG) of the features in each.
4. **Feature diversity**: we partition F_i into type-based groups and use the chi-squared and paired-t test to calculate the similarity of each pair in a group. We generate meta-features using the tests' statistic values.

Candidate feature-based meta-features. These meta-features represent the interactions of each $f_{i,j}^{cand}$ with regard to F_i. Because our candidate features are generated, our meta-features also take into account the operator(s) and features ("parent features") used to generate $f_{i,j}^{cand}$. The meta-features can be partitioned into three groups:

1. **Entropy and statistical tests for the candidate feature**: we partition F_i into subgroups based on their type and use the chi-sqaured and paired t-tests to derive statistics on $f_{i,j}^{cand}$'s correlation to each groups. In addition, we derive entropy-based measures for $f_{i,j}^{cand}$.
2. **General information on parent features and operators**: We generate statistics representing various charac-

[1] http://www.openml.org/d/37

TABLE I: The characteristics of the datasets used in the experiments

Name	Num of Data Points	% of Minority Class	Num of Features	% of Numeric Features
Heart	270	44.4%	13	46%
Horse	368	36.9%	22	31.8%
Cancer	569	37.2%	30	100%
Indian liver	585	28.6%	10	90%
Credit	690	44.4%	15	40%
Diabetes	768	34.8%	8	100%
German credit	1,000	30%	20	35%
Diabetic ret.	1,155	46.9%	19	100%
Contraceptive	1,473	22.6%	9	66.6%
Cardiography	2,126	22.1%	22	100%
Seismic bumps	2,584	6.5%	18	77%
Space	3,107	49.6%	6	100%
Wind	6,574	46.7%	14	100%
Puma_8	8,192	49.7%	8	100%
Puma_32	8,192	49.6%	32	100%
CPU_act	8,192	30.2%	21	100%
CPU_small	8,192	30.2%	12	100%
Delta elevators	9,517	49.7%	6	100%
Mammography	11,183	2.3%	6	100%
Ailerons	13,750	42.3%	40	100%
Web Data	36,974	24%	123	100%
Bank marketing	45211	11.6%	16	43.75%
Vehicle_IT	98,528	50%	100	100%
Vehicle_Norm	98,528	50%	100	100%
Poker	1,025,010	49.8%	10	100%

teristics of the parent features and the operator(s) that participated in the creation of $f_{i,j}^{cand}$. This sub-group includes the type of each parent feature, the range of possible values and the number and type of operators used.

3. **Statistical tests on parent features**: we derive statistics on the inter-correlation of the parent features of $f_{i,j}^{cand}$, as well as their correlation with the remaining features of F_i. In addition, we generate information on the used operator(s).

We use the ranking classifier C to estimate the likelihood of each $f_{i,j}^{cand} \in F_i^{cand}$ to reduce the error, based on its meta-features. We sort the candidates by this likelihood and create a ranked list $RankedF_i^{cand}$. Candidate features whose probability is below a predefined parameter $threshold_f$ are filtered.

Training the ranking classifier. To train a ranking classifier C capable of utilizing the meta-features, we produce a large labeled training set from a repository of datasets. The entire process described in this section is done "off-line".

The training data generation process is presented in Algorithm 2. The process consists of three stages: first, for each training dataset D^T we generate the set of candidate features $f_T^{cand,1}, f_T^{cand,2} ... \in F_T^{cand}$, using the same process described in Section IV-A for *one iteration*. By doing this we create a large set of candidate features that can be analyzed.

In the second stage of the process we generate the meta-features presented above for each of the candidate features. We conduct this analysis separately for each D^T. We then use the generated meta-features to create the training set for the ranking classifier.

TABLE II: The percentage of error reduction (relative to the original set of features) obtained by each of the evaluated approaches using the decision tree algorithm.

Dataset Name	initial AUC	ML_Full	ML_Fast	IG_Full	IG_Fast
Heart	0.76	40.6%	16.4%	20.1%	2.7%
Horse	0.84	39.5%	1.6%	0%	0%
Cancer	0.92	63.61%	3.2%	28.8%	-5.9%
Indian liver	0.61	31.26%	25.3%	11.8%	14.6%
Credit	0.82	31.8%	0%	4.6%	0%
Diabetes	0.73	23.9%	0%	4.9%	20.1%
German credit	0.65	11.3%	2.9%	6.8%	12.3%
Diabetic ret.	0.66	44.6%	8.2%	-9.4%	0%
Contraceptive	0.68	10.6%	0%	-2.2%	0%
Cardiography	0.89	47.5%	8.7%	-35.3%	0%
Seismic bumps	0.57	11.7%	0%	1%	0%
Space	0.8	48.4%	2.4%	13.7%	-3.3%
Wind	0.85	19.9%	0%	1.1%	0%
Puma_8	0.87	23.9%	0%	18.7%	4.7%
Puma_32	0.84	31.8%	0%	-16.2%	0%
CPU_act	0.92	23.2%	0%	-20.5%	0%
CPU_small	0.91	26%	4.5%	0.35%	-22%
Delta elevators	0.89	24.1%	0%	9%	0%
Mammography	0.82	47.51%	0%	27.8%	0%
Ailerons	0.87	30.5%	0.6%	5.1%	0%
Web Data	0.73	1.7%	0%	0.8%	0%
Bank marketing	0.72	29.5%	0%	7.5%	3.1%
Vehicle_IT	0.8	1.6%	0%	-1.6%	0%
Vehicle_Norm	0.8	11.4%	2.3%	6.1%	0.8%
Poker	0.82	57.5%	5.8%	57.5%	4.4%
Average (Median)		**29.3% (29.4%)**	**3.4% (0.3%)**	**5.6% (4.8%)**	**1.6% (0)%**

The third stage is assigning labels ("good" or "bad") to each candidate feature generated for D^T. We use the more computationally-expensive (wrapper) feature evaluation methods to determine the error reduction obtained by applying the classifier on the joint set $F_T \cup \{f_T^{cand,i}\}$ where F_T is the original feature set of D^T. If the error after adding $f_T^{cand,i}$ declines by more than ϵ, we label $f_T^{cand,i}$ as "good" and "bad" otherwise. We assign these labels to the meta-features and use the labeled set to train C.

By performing this process over a diverse group of datasets, we created a ranking classifier capable of effectively analyzing previously unseen datasets. In our experiments, presented in Section V, we demonstrate that our approach is effective for datasets with large varieties in size and feature compositions.

C. Candidate Features Evaluation & Selection

The goal of this phase is to conduct a more computationally-intensive (wrapper) evaluation on a small set of candidate features. We created this features set, $RankedF_i^{cand} \in F_i^{cand}$, during the candidate feature ranking phase.

For each candidate feature $f_{i,j}^{cand} \in RankedF_i^{cand}$, we evaluate the performance of the classifier on the joint feature set $F_i \cup \{f_{i,j}^{cand}\}$. The evaluation is conducted using k-fold cross validation. The evaluation continues until either:

- The features set $F_i \cup \{f_{i,j}^{cand}\}$ satisfies the condition: $\mathcal{E}(F_i \cup \{f_{i,j}^{cand}\}) + \epsilon_w \leq \mathcal{E}(F_i)$, where $\mathcal{E}$ measures the error and ϵ_w is a predefined parameter.
- The number of evaluated candidate features exceeded a predefined parameter R_w. In this case, the highest performing member of $RankedF_i^{cand}$ is chosen, given

TABLE III: The percentage of error reduction (relative to the original set of features) obtained by each of the evaluated approaches using the SVM algorithm.

Dataset Name	initial AUC	ML_Full	ML_Fast	IG_Full	IG_Fast
Heart	0.91	-14.3%	0%	-25.2%	-9.4%
Horse	0.83	18.3%	0%	-4%	3.5%
Cancer	0.96	46.3%	0%	0%	0%
Indian liver	0.5	9.6%	7.1%	9.6%	0%
Credit	0.85	19%	0%	-1.3%	0%
Diabetes	0.72	9.4%	0%	7.5%	0%
German credit	0.68	14.7%	2.2%	3.1%	-2.2%
Diabetic ret.	0.69	36.5%	0%	1.2%	0%
Contraceptive	0.5	25%	0%	14.6%	0%
Cardiography	0.82	41.6%	0%	9.4%	0%
Seismic bumps	0.5	1.5%	0%	0%	-3.3%
Space	0.82	27.1%	0%	13.9%	0%
Wind	0.86	12.1%	0%	7.8%	0%
Puma_8	0.82	4.2%	-1.4%	-1.6%	-0.7%
Puma_32	0.64	23.1%	0.1%	18.4%	-4.3%
CPU_act	0.90	23.6%	0%	16.7%%	9.6%
CPU_small	0.86	25.1%	0%	14.7%	0%
Delta elevators	0.89	0.8%	0%	-3.4%	0.9%
Mammography	0.53	45.7%	1.9%	32.2%	-18.5%
Ailerons	0.87	2.4%	0%	1.1%	0%
Web Data	0.74	10.3%	0%	-1.3%	0%
Bank marketing	0.67	10.3%	2.6%	4.8%	0%
Vehicle_IT	0.85	10.6%	1.1%	-1.3%	-0.5%
Vehicle_Norm	0.85	10.9%	1.3%	0.4%	0%
Poker	0.51	8.34%	0%	-3.5%	-1.1%
Average (Median)		**17.4% (14.7%)**	**0.6% (0%)**	**4.6% (1.2%)**	**-1% (0%)**

TABLE IV: The percentage of error reduction (relative to the original set of features) obtained by each of the evaluated approaches using the Random Forest algorithm.

Dataset Name	initial AUC	ML_Full	ML_Fast	IG_Full	IG_Fast
Heart	0.94	15.3%	0%	22.2%	-14.3%
Horse	0.87	26%	0%	2.6%	-3.9%
Cancer	0.99	52.4%	4.9%	-3.3%	0%
Indian liver	0.78	9.3%	3.1%	-12.7%	0%
Credit	0.91	24.8%	0%	-2.86%	2.6%
Diabetes	0.82	20.8%	0%	-1.85%	0%
German credit	0.78	16.2%	0%	1%	3.7%
Diabetic ret.	0.77	0%	1.6%	-0.1%	0%
Contraceptive	0.72	12%	0%	-9.3%	0%
Cardiography	0.95	48.1%	3.9%	-18%	0%
Seismic bumps	0.59	7.1%	7.4%	5.3%	0%
Space	0.89	46.6%	1.9%	24.4%	-3.4%
Wind	0.94	8.6%	0%	2.2%	-3%
Puma_8	0.9	8%	0%	5%	0%
Puma_32	0.94	67.7%	0.5%	49.31	0%
CPU_act	0.98	13.9%	10.2%	10%%	0.5%
CPU_small	0.97	13.6%	1.8%	1.35%	-0.6%
Delta elevators	0.94	7.6%	0%	6.5%	-0.5%
Mammography	0.96	40.6%	0%	15%	0%
Ailerons	0.94	49.6%	0%	12.2%	0%
Web Data	0.81	0.27%	0%	-2.3%	0.4%
Bank marketing	0.83	31.4%	4.1%	3.4%	0.7%
Vehicle_IT	0.91	1.3%	0%	-0.6%	0%
Vehicle_Norm	0.91	9.4%	0%	12.2%	0.3%
Poker	0.93	13.2%	8.8%	57%	5.1%
Average (Median)		**23.5% (16.2%)**	**1.9% (0%)**	**6.87% (2.2%)**	**-0.5% (0%)**

that it exceeds the minimal threshold $threashold_w$. If no candidate feature meets this criterion, no candidate feature is selected and the search terminates.

Once a candidate feature that satisfies the criteria is found, it is denoted as f_i^{select}. We then define $F_{i+1} \leftarrow F_i \cup \{f_i^{select}\}$ as the current features set for the next iteration and repeat the process described in this section

V. Evaluation

A. Experimental Setup

We evaluated ExploreKit on 25 supervised classification datasets with large variety in size, number of attributes, feature type composition and class imbalance. All datasets are available on the OpenML repository[2] and their properties are presented in Table I. All datasets were randomly partitioned into training and test sets, with 66% of the data assigned to the former. Original class ratios were maintained.

We use three types of algorithms in the evaluation: the C4.5 decision tree algorithm, SVM and random forest. For all algorithms we used the implementation included in version 3.7 of the Weka ML platform [5] with the default settings.

We used the following settings throughout the evaluation:

- We conducted our experiments on a 20-core machine with 64GB of RAM.
- We used the error reduction as the evaluation metric. We calculate it using the formula $\frac{(1-AUC_i)-(1-AUC_f)}{(1-AUC_i)}$, where AUC_i and AUC_f are the AUC values obtained for F_i and $F_i \cup \{f_{i,j}^{cand}\}$, respectively.
- For all purposes of feature generation ranking and evaluation, the training set was partitioned into three folds of equal size and class ratios. The same dataset partitions were used in all datasets.
- We used random forest to train a ranking model on the meta-features. This was done for all experiments (see Section IV-B).
- We used the following parameters: $threshold_f = 0.001$, $threshold_w \geq 0$, $\epsilon_w = 0.01$ and $R_w = 15,000$ (see Section IV for details).
- For each dataset the evaluation consisted of 15 search iterations, with one feature added during each iteration. The allowed running time for each dataset was three days – if the search exceeded that time, the last completed iteration was considered as the final one.
- We used a leave-one-out (LOO) approach for the training of the ranking classification model: for each evaluated dataset d_i, we trained the ranking classifier using meta-features from $d_j \in D$ where $i \neq j$.

B. The Evaluated Algorithms

We evaluate two versions of our algorithm and compare them to two baselines:

- **ML_Full** – the full algorithm presented in Section IV.
- **ML_Fast** – we use the candidate ranking method as in ML_Full, but evaluate only the top ranking candidate (ties are broken randomly). In this experiment the parameter ϵ_w is set to 0 (i.e. the set $F_i \cup \{f_{i,j}^{cand}\}$ needs only not to have reduced performance compared to F_i). If this condition is met, the candidate feature is added to F_i and

[2]http://www.openml.org/home

Algorithm 2 Generating metafeatures from background datasets

procedure METAFEATURES(dataset, operators, filterEvaluator, wrapperEvaluator, improvementThreshold)
  metaFeaturesSet $\leftarrow \emptyset$
  $originalPerformance \leftarrow wrapperEvaluator$.**Rank**$((dataset.GetFeatures()))$
  candidatePool $\leftarrow$ **GenerateCandidateFeatures**$(dataset$.**GetFeatures**$())$
  for each (*candidateFeature*) in *candidatePool* **do**
    candidateFeatureMetaSet $\leftarrow$ **GenerateMetaFeatures**$(candidateFeature, dataset)$
    $candidateFeaturePerformance \leftarrow wrapperEvaluator$.**Rank**$(Dataset, candidateFeature)$
    if candidateFeaturePerformance $\geq$ originalPerformance + improvementThreshold **then**
      $candidateFeatureMetaSet$.**SetLabel**$(True)$
    else
      $candidateFeatureMetaSet$.**SetLabel**$(False)$
    metaFeaturesSet $\leftarrow candidateFeatureMetaSet$
  return *metaFeaturesSet*

a new search iteration begins. If not, we terminate the search and return F_i as the final features set.

- **IG_Full** – this baseline method is identical to ExploreKit but utilizes Information Gain (IG) for the ranking of candidate features instead of our ML-based approach.
- **IG_Fast** – This baseline is identical to ML_Fast but utilizes Information Gain as the ranking function.

C. Evaluation Results

Tables II–IV show the results of the evaluation. For each classifier we compare the results of the evaluated algorithms to those obtained on the original datasets. ML_Full significantly outperforms all other methods, achieving an average error reduction of 17.4%–29.3% over the different classifiers, compared to 1.2%–5.6% for IG_Full (the closest competitor).

The results also illustrate the merits of our ML-based approach, which outperforms the IG-based methods both in ranking-only and ranking-and-evaluation settings. Another important advantage is robustness: throughout the evaluation, both ML-based variations had a negative impact on only a single dataset compared with 22 and 17 for IG_Full and IG_Fast respectively.

VI. DISCUSSION

We analyzed the meta-features used by our approach identify those that had the nost infulence on the classification model. We identified the following three groups:

- **Intial Evaluation Measures, dataset-based meta-features** – indicative meta-features of this group include statistics on the IG scores of the dataset's features and recall/precision values.
- **Feature Diversity Measures, dataset-based meta-features** – the statistical tests we used to calculate the correlation among the dataset's features (particularly the non-discrete) were given a large weight.
- **Entropy & Statistical Tests, feature-based meta-features** – specifically, the correlation of $f_{i,j}^{cand}$ with F_i, calculated using the paired T and chi-squared tests.

These findings confirm our intuition that analyzing the classifier's performance on the initial dataset as well as analyzing its feature composition are needed for effective feature selection. To the best of our knowledge, we are the first to use this information in the context of feature selection.

VII. CONCLUSIONS AND FUTURE WORK

In this study we have presented ExploreKit, a novel approach for the automatic exploration of data. By both developing a method for the generation of a large number of candidate features as well as a new type of feature selection approach, we have been able to significantly improve the performance of multiple classifiers on a large variety of datasets.

For future work, we plan to add additional operators of various types to our framework while also leveraging knowledge from the previously analyzed datasets for selecting subsets of operators for different datastes. In addition, we intend to extend our framework to include classifier and parameter selection as well as the automatic selection and application of methods such as PCA [2]. Finally, we are interested in combining our approach with deep learning algorithms.

VIII. ACKNOWLEDGMENT

This work was supported in part by DARPA under the award DARPA FA8750-15-2-0104.

REFERENCES

[1] J. Dougherty, R. Kohavi, M. Sahami, et al. Supervised and unsupervised discretization of continuous features. In *Machine learning: proceedings of the twelfth international conference*, volume 12, 1995.
[2] G. H. Dunteman. *Principal components analysis*, volume 69. Sage, 1989.
[3] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada. Normalized mutual information feature selection. *Neural Networks, IEEE Transactions on*, 20(2), 2009.
[4] S. Garcia, J. Luengo, J. A. Sáez, V. López, and F. Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *Knowledge and Data Engineering, IEEE Transactions on*, 25(4), 2013.
[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1), 2009.
[6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
[7] J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*. IEEE, 2015.
[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3), 2015.