Explaining Inference Queries with Bayesian Optimization

Brandon Lockhart[◇], Jinglin Peng[◇], Weiyuan Wu[◇], Jiannan Wang[◇], Eugene Wu[†]
Simon Fraser University[◇] Columbia University[†]
{brandon_lockhart, jinglin_peng, youngw, jnwang}@sfu.ca ewu@cs.columbia.edu

ABSTRACT

Obtaining an explanation for an SQL query result can enrich the analysis experience, reveal data errors, and provide deeper insight into the data. Inference query explanation seeks to explain unexpected aggregate query results on inference data; such queries are challenging to explain because an explanation may need to be derived from the source, training, or inference data in an ML pipeline. In this paper, we model an objective function as a black-box function and propose BOExplain, a novel framework for explaining inference queries using Bayesian optimization (BO). An explanation is a predicate defining the input tuples that should be removed so that the query result of interest is significantly affected. BO — a technique for finding the global optimum of a black-box function — is used to find the best predicate. We develop two new techniques (individual contribution encoding and warm start) to handle categorical variables. We perform experiments showing that the predicates found by BOExplain have a higher degree of explanation compared to those found by the state-of-the-art query explanation engines. We also show that BOExplain is effective at deriving explanations for inference queries from source and training data on a variety of real-world datasets. BOExplain is open-sourced as a Python package at https://github.com/sfu-db/BOExplain.

PVLDB Reference Format:
Brandon Lockhart, Jinglin Peng, Weiyuan Wu, Jiannan Wang, Eugene Wu. Explaining Inference Queries with Bayesian Optimization. PVLDB, 14(11): 2576 - 2585, 2021.
doi:10.14778/3476249.3476304

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sfu-db/BOExplain.

1 INTRODUCTION

Data scientists often need to execute aggregate SQL queries on *inference data* to inspect a machine learning (ML) model's performance. We call such queries *inference queries*, which can be seen as an SQL query whose expressions may perform model inference. Consider an inference dataset with four variables (customer_id, age, sex, M.predict(I)), where M.predict(I) represents a variable where each value denotes whether the model M predicts the customer will be a repeat or one-time buyer. Running the following inference query will return the number of female (predicted) repeat buyers:

```
SELECT COUNT(*) FROM InferenceData as I
WHERE sex = 'female' and M.predict(I) = 'repeat'
```

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476304

Table 1: Comparison of BOExplain and existing approaches.

	SQL Explain [1, 40, 41, 52]	Inference Query Explain	
		Rain [53]	BOExplain
Inference Data	Supported	Supported	Supported
Training Data	Not Supported	Supported	Supported
Source Data	Not Supported	Not Supported	Supported
Explanation Type	Coarse-grained	Fine-grained	Coarse-grained
Methodology	White-box	White-box	Black-box

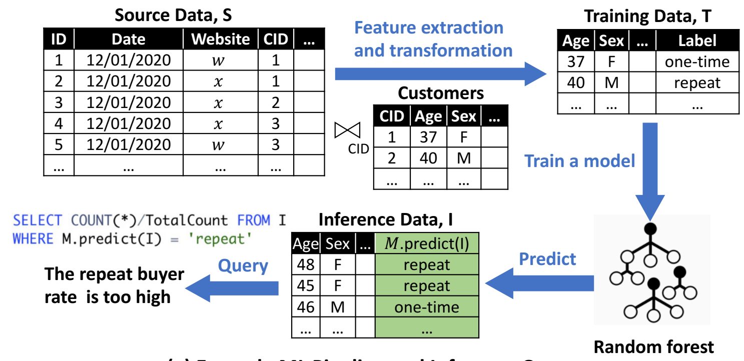
If the query result is surprising, e.g., the number of repeat buyers is higher than expected, the data scientist may seek an explanation. One popular explanation method is to find a subset of the input data such that when this subset is removed, and the query is re-executed, the unexpected result no longer manifests [41, 52]. This method is known as a *provenance* or *intervention*-based explanation [34].

There are two types of explanations in the intervention-based setting: fine-grained (a set of tuples) and coarse-grained (a predicate) [33]. This paper focuses on coarse-grained explanations. Predicates, unlike sets of tuples, provide comprehensible explanations and identify common properties of the input tuples that cause the unexpected result. For the above example, it may return a predicate like `sex = 'female' AND 20` $\leq$ `age` $\leq$ `25` which suggests that if the young female customers are removed from the inference data, the query result will look normal. Then, the data scientist can inspect these customers and conduct further investigation.

Generating an explanation from inference data can certainly help to understand the answer to an inference query. However, an ML pipeline does not only contain inference data but also training and source data. The following example illustrates a scenario where an explanation should be generated from source data.

Example 1.1. CompanyX creates an ML pipeline (Figure 1(a)) to predict repeat customers for a promotional event. CompanyX receives transaction records from several websites that sell their products and aggregates them into a source data table S. Next, the user defined function (UDF) make_training(·) extracts and transforms features into the training dataset T. Finally, a random forest model is fit to the training data, and the model is applied to the inference dataset I which updates it with a prediction variable, $M.predict(I)$.

For validation purposes, the data scientist writes a query to compute the percentage of repeat buyers. The rate is higher than expected, but she wants to double check that the result is not due to a data error. In fact, it turns out that the source data S contains errors during `Date` $\in [t_1, t_2]$*, when the website w had network issues; customers confirmed their transactions multiple times, which led to duplicate records in S. The training data extraction UDF was coded to label customers with multiple purchases as repeat buyers, and labelled all of the w customers during the network issue as repeats. The model erroneously predicts every website w customer as a repeat buyer, and thus leads to the high query result. Ideally, the data scientist could ask whether the source data contains an error, and an explanation system would generate a predicate ($t_1 \leq$* `Date` *$\leq t_2$* `AND Website` *$= w$).*



ID	Date	Website	CID	...
1	12/01/2020	*w*	1	
2	12/01/2020	*x*	1	
3	12/01/2020	*x*	2	
4	12/01/2020	*x*	3	
5	12/01/2020	*w*	3	
...	...	...	...	

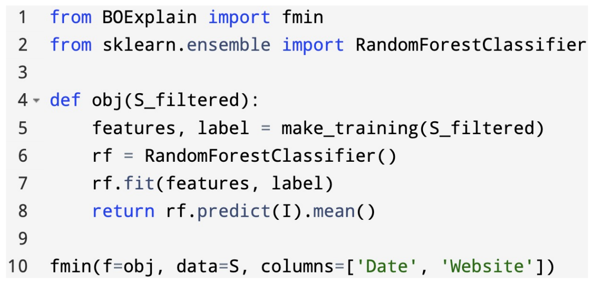
CID	Age	Sex	...
1	37	F	
2	40	M	
...	...	...	

Age	Sex	...	Label
37	F		one-time
40	M		repeat
...	...		...

Age	Sex	...	*M*.predict(I)
48	F		repeat
45	F		repeat
46	M		one-time
...	...		...

```
SELECT COUNT(*)/TotalCount FROM I
WHERE M.predict(I) = 'repeat'
```

(a) Example ML Pipeline and Inference Query



```
from BOExplain import fmin
from sklearn.ensemble import RandomForestClassifier

def obj(S_filtered):
    features, label = make_training(S_filtered)
    rf = RandomForestClassifier()
    rf.fit(features, label)
    return rf.predict(I).mean()

fmin(f=obj, data=S, columns=['Date', 'Website'])
```

(b) Inference Query Explanation from Source Data Using BOExplain

Figure 1: An illustration of using BOExplain to generate an explanation from source data in an ML pipeline.

Unfortunately, existing SQL explanation approaches [1, 40, 41, 52] are ill-equipped to address this setting (Table 1) because they are based on analysis of the query provenance. Although they can generate a predicate explanation over the inference data, the provenance analysis does not extend across model training nor UDFs, which are prevalent in data science workflows. The recent system Rain [53] generates fine-grained explanations for inference queries. It relaxes the inference query into a differentiable function over the model's prediction probabilities, and leverages influence analysis [25] to estimate the query result's sensitivity to a training record. However, Rain returns training records rather than predicates, and estimating the model prediction sensitivity to group-wise changes to the training data is an open problem. Further, Rain does not support UDFs and uses a white-box approach that is less amenable to data science programs (Figure 1(b)) that heavily incorporate UDFs.

As a first approach towards addressing the above limitations, and to diverge from existing white-box explanation approaches [1, 40, 41, 52, 53], this paper explores a black-box approach for inference query explanation. BOExplain models inference query explanation as a hyperparameter tuning problem and adopts Bayesian Optimization (BO) to solve it. In ML, hyperparameters (e.g., the number of trees, learning rate) control the training process and are tuned in an "outer-loop" that surrounds the training process. Hyperparameter tuning seeks to find the best hyperparameters that maximizes some model quality measure (e.g., validation score). BOExplain treats predicate constraints (e.g., t_1, t_2, w in Example 1.1) as hyperparameters, and aims to assign the optimal value to each constraint. By defining a metric that evaluates a candidate explanation's quality (e.g., the repeat buyer rate decrease), BOExplain finds the constraint values that correspond to the highest quality predicate.

A black-box approach offers many advantages for inference query explanation. In terms of **usability**, a data scientist can derive a predicate from any data involved in an ML pipeline rather than inference data only. Furthermore, its concise API design is similar to popular hyperparameter tuning libraries, such as scikit-optimize [22] and Hyperopt [9], that many data scientists are already very familiar with. Figure 1(b) shows an example using BOExplain's API to solve Example 1.1. The data scientist wraps the portion of the program in an objective function obj whose input is the dataset to generate predicates for, and whose output is the repeat buyer rate that should be minimized. She also provides hints to focus on the Date and Website variables. See Section 3.2 for more details.

In terms of **adaptability**, a black-box approach can potentially be used to generate explanations for any data science workflow beyond inference queries. The current ML and analytics ecosystem is rapidly evolving. In contrast to white-box approaches, which must be carefully designed for specific programs, BOExplain can more readily evolve with API, library, model, and ecosystem changes.

In terms of **effectiveness**, BOExplain builds on the considerable advances in BO by the ML community [48], to quickly generate high quality explanations. A secondary benefit is that BO is a progressive optimization algorithm, which lets BOExplain quickly propose an initial explanation, and improve it over time.

The key technical challenge is that existing BO approaches [10, 23, 50] cannot be naively adapted to explanation generation. In the hyperparameter tuning setting, categorical variables typically have very low cardinality (e.g., with 2-3 distinct values [35]). In the query explanation setting, however, a categorical variable can have many more distinct values. To address this, we propose a categorical encoding method to map a categorical variable into a numerical variable. This lets BOExplain estimate the quality of the categorical values that have not been evaluated. We further propose a warm start approach so that BOExplain can prioritize predicates with more promising categorical values.

In summary, this paper makes the following contributions:

- We are the first to generate coarse-grained explanations from the training and source data to an inference query. We argue for a black-box approach to inference query explanation and discuss its advantages over a white-box approach.
- We propose BOExplain, a novel query explanation framework that derives explanations for inference queries using BO. We develop two techniques (categorical encoding and warm start) to improve BOExplain's performance on categorical variables.
- We show that BOExplain can generate comparable or higher quality explanations than state-of-the-art SQL explanation engines (Scorpion [52] and MacroBase [1]) on SQL-only queries. We evaluate BOExplain using inference queries on real-world datasets showing that BOExplain can generate higher quality explanations than random search for various input datasets.

2 PROBLEM DEFINITION

In this section, we first define the SQL explanation problem, and subsequently describe the extension to inference query explanation.

2.1 Background: SQL Explanation

Query. In this work, we focus on aggregation queries over a single table (the extension to multiple tables has been formalized in [41]). An *explainable query* is an arithmetic expression over a collection of SQL query results, as formally defined in Definition 1.

Definition 1 (Supported Queries)**.** *Given a relation R, an explainable query $Q = E(q_1, \ldots, q_k)$ is an arithmetic expression E over queries $q_1, \ldots, q_k$ of the form*

$$q_i = \textbf{SELECT}\ agg(\ldots)\ \textbf{FROM}\ R$$
$$\textbf{WHERE}\ C_1\ \textbf{AND/OR}\ \ldots\ \textbf{AND/OR}\ C_m$$

where agg is an aggregation operation and C_j is a filter condition.

Example 2.1. Returning to the running example in Section 1, the user queries the predicted repeat buyer rate. This can be expressed as $Q = q_1/q_2$, an arithmetic expression over q_1 and q_2 where

q_1 = *SELECT COUNT(*) FROM I WHERE M.predict(I)='repeat';*

q_2 = *SELECT COUNT(*) FROM I;*

Complaint. After the user executes a query, she may find that the result is unexpected and *complain* about its value. In this work, the user can complain about the result being too high or too low, as done in [41]. We use the notation $dir = low$ ($dir = high$) to indicate that Q is unexpectedly high (low).

Example 2.2. In our running example, the user found the repeat buyer rate too high. Thus along with the query Q from Example 2.1, the user specifies $dir = low$ to indicate that Q should be lower.

Explanation. After the user complains about a query result, BOExplain will return an explanation for the complaint. In this work, we define an explanation as a predicate over given variables.

Definition 2 (Explanation). *Given numerical variables $N_1, \ldots, N_n$ and categorical variables $C_1, \ldots, C_m$, an explanation is a predicate p of the form*

$$p = l_1 \leq N_1 \leq u_1 \wedge \cdots \wedge l_n \leq N_n \leq u_n \wedge C_1 = c_1 \wedge \cdots \wedge C_m = c_m.$$

The set of all such predicates forms the predicate space S.

Example 2.3. The source data in Figure 1 contains the variables Date and Website. An example explanation over these variables is $12/01/2020 \leq Date \leq 12/10/2020 \wedge Website = w$.

Objective Function. Next we define our objective function. The goal of our system is to find the best explanation for the user's complaint. Hence, we need to measure the quality of an explanation. For a predicate p, let $\sigma_{\neg p}(R)$ represent R filtered to contain all tuples that do not satisfy p. We apply the query to $\sigma_{\neg p}(R)$ and get the new query result. If the user specifies $dir = low$, then the smaller the new query result is, the better the explanation is. Hence, we use the new query result as a measure of explanation quality. The objective function is formally defined in Definition 3.

Definition 3 (Objective Function). *Given a predicate p, relation R, and query Q, the objective function $obj(p, R, Q) \rightarrow \mathbb{R}$ applies Q on the relation $\sigma_{\neg p}(R)$.*

With the definition of objective function, the problem of searching for the best explanation is equivalent to finding a predicate that minimizes or maximizes the objective function.

Definition 4 (SQL Explanation Problem). *Given a relation R, query Q, direction dir, and predicate space S, find the predicate $p^* = \arg\min_{p \in S} obj(p, R, Q)$ if $dir = low$ (use $\arg\max$ if $dir = high$).*

It may appear that minimizing the above objective function runs the risk of overfitting to the user's complaint (perhaps with an overly complex predicate). However, a regularization term can be placed within the objective function—for instance, SQL explanation typically regularizes using the number of tuples that satisfy the predicate [52]. Since Q is an arithmetic expression over multiple queries, one of those queries may simply be the regularization term.

2.2 Extension to Inference Query Explanation

For inference query explanation, we focus on three input datasets that the user can generate explanations from: source, training, and inference. The query processing pipeline is as follows (Figure 1(a)):

(1) Transform and featurize the source data into the training data.
(2) Train an ML model over the training data.
(3) Use the model to predict a variable from the inference dataset.
(4) Issue a query over the inference dataset.

From the above workflow, we can find that there are two differences between SQL and inference query explanations: 1) the query for inference query explanation is evaluated on the inference data with model predictions, and 2) in inference query explanation, the user may want an explanation for the input dataset at any step of the workflow (e.g., the source, training, or inference dataset), while SQL explanation only consider the query's direct input.

We next formally define the scope of the errors that we seek to explain in Definition 5.

Definition 5 (Scope of Errors). *This paper focuses on errors in the form of systematically mislabelled tuples that can be described using a predicate as defined in Definition 2.*

We next extend the objective function from SQL explanation to inference query explanation. Let Q be the query issued by the user over the updated inference data, with the same form as in Definition 1. Let R be the data that we want to derive an explanation from (it can be source, training, or inference data) and p be an explanation (i.e., predicate) over R. We measure the quality of p like in SQL explanation: filter the data by p, then get the new query result. Note that for inference query explanation, the query is issued over the updated inference data. Hence, we define $\mathcal{P}$ as the subset of the ML pipeline that takes as input the dataset R that we wish to generate an explanation from, and that outputs the updated inference data which is used as input to the SQL query. The extended objective function is defined in Definition 6.

Definition 6 (Objective Function). *Given a subset of an ML pipeline $\mathcal{P}$, a predicate p, relation R, and query Q, the objective function $obj(p, R, \mathcal{P}, Q) \rightarrow \mathbb{R}$ feeds $\sigma_{\neg p}(R)$ through $\mathcal{P}$, and then applies Q on the inference data.*

Finally, we define the inference query explanation problem.

Definition 7 (Inference Query Explanation Problem). *Given a relation R, query Q, direction dir, pipeline $\mathcal{P}$, and predicate space S, find the predicate $p^* = \arg\min_{p \in S} obj(p, R, Q, \mathcal{P})$ if $dir = low$ (use $\arg\max$ if $dir = high$).*

We assume that an explanation in the form of Definition 2 that performs well under the objective function in Definition 7 is meaningful to the user. Hence, if $dir = low$ ($high$), the predicate p^* that minimizes (maximizes) the objective function is considered optimal.

3 THE BOEXPLAIN FRAMEWORK

This section introduces Bayesian optimization (BO) and presents the BOExplain framework.

3.1 Background

Black-box optimization aims to find a global minimum (or maximum) $x^* = \min_{x \in \mathcal{X}} f(x)$ of a black-box function f over a search space $\mathcal{X}$. BO is a sequential model-based optimization strategy to solve the problem, where *sequential* means it is an iterative algorithm and *model-based* means it estimates f using surrogate models.

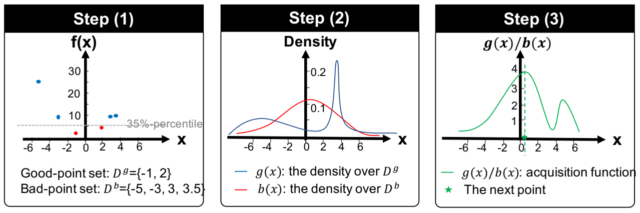


Figure 2: Suppose TPE has observed six points $D = \{(-5, 25)$, $(-3, 9)$, $(-1, 1)$, $(2, 4)$, $(3, 9)$, $(3.5, 9.25)\}$. This figure illustrates how TPE finds the next point to evaluate ($\gamma = 35\%$).

Table 2: An illustration of parameter creation.

Age	Sex	City	State	Occupation	M.predict(I)
48	F	Mesa	AZ	Athlete	repeat
45	F	Miami	FL	Artist	repeat
46	M	Mesa	AZ	Writer	one-time
40	M	Miami	FL	Athlete	repeat
42	F	Miami	FL	Athlete	repeat

Tree-structured Parzen Estimator (TPE). TPE [8, 10] is a popular BO algorithm. It first initializes by evaluating f on random samples from the search space. Then, it iteratively selects x from the search space using an *acquisition function* and evaluates $f(x)$. Let $D = \{(x_1, f(x_1)), \cdots, (x_t, f(x_t))\}$ be the set of samples evaluated in previous iterations. TPE chooses the next sample as follows:

(1) Partition D into sets D^g and D^b, where D^g consists of the set of γ-percentile points with the lowest $f(x)$ values in D, and D^b consists of the remaining points (γ is a user-definable parameter). Since the goal is minimize $f(x)$, D^g is called the *good-point set* and D^b is called the *bad-point set*. Intuitively, good points lead to smaller objective values than bad points.
(2) Use Parzen estimators (a.k.a kernel density estimators) to build a density model $g(x)$ and $b(x)$ over D^g and D^b, respectively. Intuitively, given an unseen x^* in the search space, the density models $g(x^*)$ and $b(x^*)$ can return the probability of x^* being a good and bad point, respectively. Note that separate density models $g(x)$ and $b(x)$ are constructed for each dimension of $\mathcal{X}$.
(3) Construct an acquisition function $g(x)/b(x)$ and select x with the maximum $g(x)/b(x)$ to evaluate in the next iteration. Intuitively, TPE selects a point that is more likely to appear in the good-point set and less likely to appear in the bad-point set.

Figure 2 illustrates the three steps. Please refer to our technical report [30] for a complete introduction to TPE.

Categorical Variables. TPE models categorical variables by using categorical distributions rather than kernel density estimation. Consider a categorical variable with four distinct values: `Website` $\in \{w_1, w_2, w_3, w_4\}$. To build $g($`Website`$)$, TPE estimates the probability of w_i based on the fraction of its occurrences in D^g; the distribution is smoothed by adding 1 to the count of occurrences for each value. For instance, if the occurrences are 2, 0, 1, 0, then the distribution $g($`Website`$)$ is $\{P(w_1), P(w_3), P(w_3), P(w_4)\} = \{3/7, 1/7, 2/7, 1/7\}$.

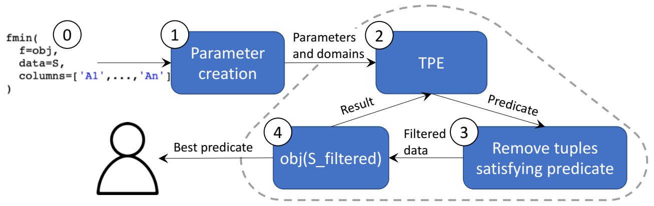
3.2 Our Framework

In this section, we describe the BOExplain framework.

Parameter Creation. Given a predicate space, we need to map it to a parameter search space (the parameters and their domains). Suppose a predicate space is defined over variables $A_1, A_2, \cdots, A_n$.

If A_i is numerical (e.g., age, date), two parameters are created that serve as bounds on the range constraint. Specifically, the parameters $A_{i_{\text{min}}}$ and $A_{i_{\text{length}}}$ define the lower bound and the length of the range constraint, respectively. $A_{i_{\text{min}}}$ and $A_{i_{\text{length}}}$ have interval domains $[\min(A_i), \max(A_i)]$ and $[0, \max(A_i) - \min(A_i)]$, respectively.

If A_i is categorical (e.g., sex, website), one categorical parameter is created with a domain consisting of all unique values in A_i.



```
fmin(
  f=obj,
  data=S,
  columns=['A1',...,'An']
)
```

Figure 3: The BOExplain framework.

Example 3.1. Suppose the user defines a predicate space over `State` *and* `Age` *in Table 2. BOExplain creates three parameters: one categorical parameter for* `State` *with domain {AZ, FL}, and two numerical parameters for* `Age` *with domains* $[40, 48]$ *and* $[0, 8]$*, respectively.*

BOExplain Framework. Figure 3 walks through the BOExplain framework. In step ⓪, the user provides an objective function obj, a relation S, and predicate variables $A_1, \ldots, A_n$ (Figure 1(b), line 10). Step ① creates the parameters and their domains. Step ② runs one iteration of TPE, starting with the parameters from step ①, and outputs a predicate. Steps ③ and ④ evaluate the predicate by removing those tuples from the input dataset, and evaluating obj on the filtered data. The result is passed to TPE for the next iteration, and possibly yielded to the user as an intermediate or final predicate explanation.

Consider the example code in Figure 1(b). Once it is executed, BOExplain first creates three parameters: `Date`$_{\text{min}}$, `Date`$_{\text{length}}$, and `Website` along with corresponding domains. Then, it iteratively calls TPE to propose predicates (e.g., "`12/01/2020` $\leq$ `Date` $\leq$ `12/02/2020` AND `Website` $= w$"). BOExplain obtains S_filtered by removing the tuples that satisfy this predicate from S. Next, it applies `obj`$(\cdot)$ to S_filtered which will rerun the pipeline (Figure 1(a)) to compute the updated repeat buyer rate. The predicate and the updated rate are passed to TPE to use when selecting the predicate on the next iteration. This iterative process repeats until the time budget is reached. When the user stops BOExplain, or when the optimization has converged, the predicate with the lowest rate is returned.

Why Is TPE Suitable For Query Explanation? Recent work [7, 28, 32] has suggested that random search is a competitive hyperparameter tuning strategy for various ML tasks. However, we find that TPE is more effective for query explanation because it is designed for problems where similar parameter values tend to have similar objective values (e.g., model accuracy). TPE can leverage this property to prune poor regions of the search space. As a trivial example, suppose a hyperparameter controls the number of trees in a random forest. If values 10, 12, 14 have resulted in a poor objective value, then TPE will down-weigh similar values (e.g., 9, 16).

This property tends to hold in query explanation, because similar predicates tend to have similar objective values. For instance, we would expect that the predicate age $\in [10, 20]$ will exhibit a similar objective to age $\in [10, 19]$ and age $\in [10, 21]$; when the former has a poor objective value, the latter two may be pruned. If this property does not hold, BOExplain can still find the optimal predicate via the *exploration* component of BO. BO balances two components for selecting a point to evaluate: 1) *exploration* of the search space, and 2) *exploitation* of points similar to previously well-performing points. *Exploitation* may be ineffective if similar predicates do not perform similarly under the objective function, but *exploration* will still test unpromising predicates, thus eventually leading BO to the optimal predicate.

4 SUPPORTING CATEGORICAL VARIABLES

In this section, we present our techniques to enable BOExplain to support categorical variables more effectively.

4.1 Individual Contribution Encoding

Recall from Section 3.1 that TPE models numerical and categorical variables using kernel density estimation (KDE) and categorical distributions, respectively. The advantage of KDE over a categorical distribution is that it can estimate the quality of unseen points using the points that are close to them. To benefit from this advantage, we map a categorical variable to a numerical variable. We call this idea *categorical encoding*. In the following, we present our categorical encoding approach, called individual contribution (IC) encoding.

A good encoding method should put *similar* categorical values close to each other. Intuitively, two categorical values are similar if they have a similar contribution to the objective function value. Based on this intuition, we rank the categorical values by their individual contribution to the objective function value. Specifically, consider a categorical variable C with domain $\{c_1, \ldots, c_n\}$. For each value c_i, we obtain the filtered dataset $\sigma_{C \neq c_i}(S)$ w.r.t. the predicate $C = c_i$. Next, the objective function is evaluated on the relation $\sigma_{C \neq c_i}(S)$ which returns a number. This number can be interpreted as the contribution of the categorical value on the objective function. After repeating for all values c_i, the categorical values are mapped to consecutive integers in order of their IC. BOExplain will then use a numerical rather than categorical variable to model C.

Example 4.1. Suppose we would like an explanation from the inference data in Table 2. Suppose the objective function value is the repeat buyer rate and the predicate space is defined over the Occupation variable. Note that the Occupation variable has the domain {Athlete, Artist, Writer}. The IC of Athlete is determined by removing the tuples where Occupation="Athlete" and computing the objective function on the filtered dataset, which gives 0.5 *(since only one of the two tuples in the filtered dataset is a repeat buyer). Similarly, the ICs of Artist and Writer are* 0.75 *and* 1 *respectively. Finally, we sort the categorical values by their objective function value and encode the values as integers: Athlete* $\longrightarrow$ 1*, Artist* $\longrightarrow$ 2*, Writer* $\longrightarrow$ 3.

4.2 Warm Start

We next propose a warm-start approach to further enhance BOExplain's performance for categorical variables. Since an IC score has been computed for each categorical value, we can prioritize predicates that are composed of well performing individual categorical values. Rather than selecting n_{init} points at random to initialize the TPE algorithm, we select the n_{init} combinations of categorical values with the best combined score. More precisely, for a variable C_i, we consider the tuple pairs (variable value, IC) as computed in Section 4.1, $S^i_{IC} = \{(c_j, IC(c_j))\}_{j=1}^{n_i}$, where n_i is the number of unique values in variable C_i. Next, we compute the d−ary Cartesian product and add the ICs for each combination $S_{IC} = S^1_{IC} \times \cdots \times S^d_{IC} = \{((c_{i_1}, \ldots, c_{i_d}), IC(c_{i_1}) + \cdots + IC(c_{i_d})) \mid i_j \in \{1, \ldots, n_j\}\}$.

Example 4.2. The IC for values in the Occupation variable were computed in Example 4.1, $S^{\text{Occupation}}_{ic} = \{(\text{Athlete}, 0.5), (\text{Artist}, 0.75), (\text{Writer}, 1)\}$, and for Sex we have $S^{\text{Sex}}_{ic} = \{(\text{F}, 0.5), (\text{M}, 1)\}$. Next we compute the combined IC score for each combination of predicates $S_{IC} = \{((\text{Athlete}, \text{F}), 1), \ldots, ((\text{Writer}, \text{M}), 2)\}$.

To see why adding ICs can be useful for prioritizing good predicates, suppose we want to minimize the objective function, and that $C_1 = c_1$ and $C_2 = c_2$ have small ICs. Then it is likely that $C_1 = c_1 \wedge C_2 = c_2$ has a small value. So we choose to sum the IC values as it encodes this property. Finally, we select n_{init} valid predicates with the best combined IC score. Recall the user defines the direction that the objective function should be optimized. Therefore, we select the predicates with the smallest (largest) IC score if the objective function should be minimized (maximized). If the predicate also contains numerical variables, values are selected at random to initialize the range constraint parameters.

Example 4.3. Continuing with Example 4.2, recall that we want to minimize the objective function, so the smaller the combined IC score the better. Suppose $n_{\text{init}} = 2$, then on the first and second iterations of BO, we evaluate the predicates `Occupation` = "Athlete" ∧ `Sex` = "F" and `Occupation` = "Artist" ∧ `Sex` = "F" respectively. Note that `Occupation` = "Athlete" ∧ `Sex` = "F" is the best predicate, so adding IC scores can prioritize good explanations.

5 EXPERIMENTS

Our experiments seek to answer the following questions. (1) How does BOExplain compare to current state-of-the-art query explanation engines for numerical variables? (2) Are the IC encoding and warm start heuristics effective? (3) How effective is BOExplain at deriving explanations from source and training data? (4) Can BOExplain generate useful explanations for real corrupted datasets?

5.1 Experimental Settings

5.1.1 Baselines. For SQL-only queries, we compare BOExplain with the explanation engines Scorpion [52] and MacroBase [1, 5] which return predicates as explanations. For inference queries, no predicate-based explanation engines exist, so we compare with a random search baseline [7] Hyperband [27].

Scorpion [52] is a framework for explaining group-by aggregate queries. The authors define a function to measure the quality of a predicate, which can be implemented as BOExplain's objective function. Each continuous variable's domain is split into 15 equi-sized ranges as set in the original paper. We use the author's open-source code[1] to run the Scorpion experiments.

MacroBase [5] (later, the DIFF operator [1]) is an explanation engine that considers combinations of variable-values pairs, similar to a `CUBE` query [21], as candidate explanations. In Section 2.3 of [1], the authors describe how to use the DIFF operator with Scorpion's objective function. We implemented it using the author's open-source code[2]. The user needs to discretize numerical variables; we tuned the bin size from 2 to 15 and report the best result.

In [1], MacroBase was shown to outperform other explanation engines including Data X-ray [51] and Roy and Suciu [40], and so we do not compare with these approaches.

Random search is a competitive method for hyperparameter tuning [7]. The parameters are chosen independently and uniformly at random from the domains described in Section 3.2.

Hyperband [27] is an exploration-based optimization strategy that speeds up random search through adaptive resource allocation and early-stopping.

5.1.2 Real-world Datasets. The following lists the five real-world datasets used in our experiments. For House and Credit, we inject synthetic errors. An explanation is derived from source data for Credit and Amazon, training data for House and German, and inference data for NYC.

House price prediction [14]. This data was published already split into training (1460 rows) and inference (1459 rows) datasets. It

[1]https://github.com/sirrice/scorpion
[2]https://github.com/stanford-futuredata/macrobase

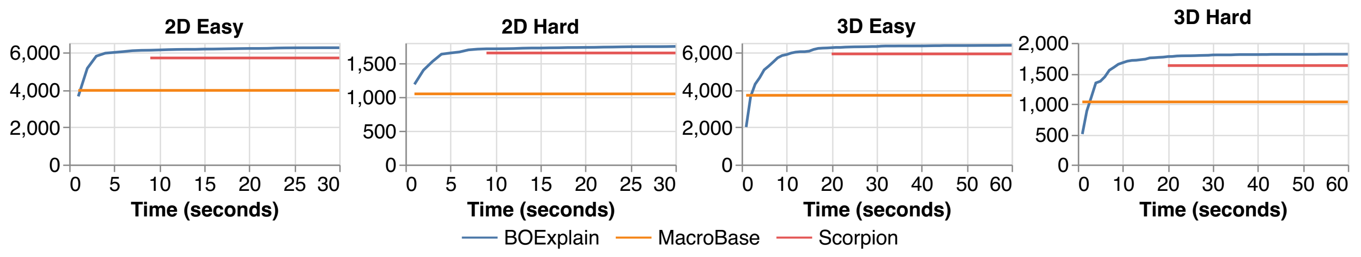


Figure 4: Performance comparison with Scorpion and MacroBase. The goal is to maximize the objective function.

contains 79 variables of a house which are used to train a support vector regression model to predict the house price.

Credit card approval prediction[3]. The source data consists of two tables: `application_record` (438,557 rows, 18 variables), which contains information about previous applicants, and `credit_record` (1,048,575 rows, 3 variables), which stores the applicants' credit history. A decision tree classifier is trained to predict whether a customer will default on their credit card payment. We set aside 20% of the data to use for the inference query, and 80% for training.

Amazon product reviews [38]. This dataset contains 6928 reviews of Amazon products with ground truth and crowdsourced binary labels. 80% of the reviews are used for training with labels formed from the majority vote of the crowdsourced labels, and 20% for testing with the ground truth labels. We encode the reviews using Count Vectorization and train a support vector classifier.

German credit risk [16]. This dataset contains 19 variables of 1000 bank customers with each customer labelled as having good or bad credit risk. We one-hot encode the categorical variables, do an 80-20 train-test split, and train an XGBoost classifier.

NYC yellow taxi dataset[4]. This dataset contains taxi trip information for every yellow taxi trip in New York City. Following the setup in [4], we predict the hourly demand by region for the 20 most frequent regions, and use the features weekday, region, demand of previous 24 hours, and cosine/sine features to encode that hours are cyclical. We train an XGBoost regressor on data from January and February, 2020, and perform inference on March 2020's data.

5.1.3 Metrics. To measure the quality of an explanation, we plot the best objective function value achieved by each time point. For Scorpion and MacroBase we plot, the objective function value corresponding to their output predicate as a line that begins when the system finishes. To evaluate the effectiveness at identifying data errors, we measure the F-score, precision, and recall in the experiments on House and Credit. We synthetically corrupt data defined by a predicate, and use that data as ground truth. Precision is the number of selected corrupted tuples divided by the total number of selected tuples. Recall is the number of selected corrupted tuples divided by the total number of corrupted tuples. F-score is the harmonic mean of precision and recall. For BOExplain, Random, and Hyperband, each result is averaged over 10 runs.

5.1.4 Implementation. BOExplain was implemented in Python 3.9. We modify the TPE algorithm in the Optuna library [2] with our optimization for categorical variables. The ML models in Section 5.3 are created with sklearn. The experiments were run single-threaded on a MacBook Air (OS Big Sur, 8GB RAM). In the TPE algorithm, we set $n_{init} = 10$, $n_{ei} = 24$, and $\gamma = 0.1$ for all experiments.

5.2 Explaining SQL-Only Queries

To compare BOExplain, Scorpion, and MacroBase, we replicate the experiment from Section 8.3 of Scorpion's paper [52], using the same datasets, query, and objective function. Note that MacroBase explicitly aims to optimize Scorpion's objective function, as described in Section 2.2 of [1]. The dataset consists of a single *group by* variable A_d, an aggregate variable A_v, and search variables $A_1, \dots, A_n$ with domain$(A_i) = [0, 100] \subset \mathbb{R}, i \in [n]$. A_d contains 10 unique values (or 10 groups) each corresponding to 2000 tuples randomly distributed in the n dimensions. 5 groups are outlier groups and the other 5 are holdout groups. Each A_v value in a holdout group is drawn from $\mathcal{N}(10, 10)$. Outlier groups are created with two n dimensional hyper-cubes over the n variables, where one is nested inside the other. The inner cube contains 25% of the tuples and $A_v \sim \mathcal{N}(\mu, 10)$, and the outer cube contains 25% of the tuples in the group and $A_v \sim \mathcal{N}((\mu + 10)/2, 10)$, else $A_v \sim \mathcal{N}(10, 10)$. μ is set to 80 for the "easy" setting (the outliers are more pronounced), and 30 for the "hard" setting (the outliers are less pronounced). The query is SELECT SUM(A_v) FROM synthetic GROUP BY A_d. The arithmetic expression over the SQL query is defined in Section 3 of [52] that forms an objective function to be maximized. The penalty $c = 0.2$ was used to penalize the number of tuples removed as described in Section 7 of [52]. We used $n = 2$ and $n = 3$ since 3 is the maximum number of variables supported by MacroBase.

The results are shown in Figure 4. BOExplain outperforms Scorpion and MacroBase in terms of optimizing the objective function in each experiment. This is because BOExplain can refine the constraint values of the range predicate which enables it to outperform Scorpion and MacroBase which discretize the range. The results are the same in the easy and hard settings. MacroBase performs poorly because the predicates formed by discretizing the variable domains into equi-sized bins, and computing the cube, do not optimize this objective function. This exemplifies a known limitation of MacroBase that binning continuous variables is difficult [1].

BOExplain also outperforms Scorpion in terms of running time. BOExplain achieves Scorpion's objective function value in around half the time on each experiment.

Note. The focus of this paper is *not* on SQL-only queries, thus we did not conduct a comprehensive comparison with Scorpion and MacroBase. This experiment aims to show that a black-box approach (BOExplain) can even outperform white-box approaches (Scorpion and MacroBase) for SQL-only queries in some situations.

5.3 Explaining Inference Queries

In this section, we evaluate BOExplain's efficacy at explaining inference queries from training and source data. In Section 5.3.1, we investigate BOExplain's approach for categorical variables on House (training data), and in Section 5.3.2, we evaluate BOExplain in a complex ML pipeline on Credit (source data).

5.3.1 **Supporting Categorical Variables.** In this experiment, we assess BOExplain's method for handling categorical variables on House. The data is corrupted by setting the tuples satisfying `Neighbourhood="CollgCr"` $\wedge$ `Exterior1st="VinylSd"` $\wedge$ $2000 \leq$ `YearBuilt` ≤ 2010 to have their sale price multiplied by 10, affecting 6.16% of the data. We query the average predicted house price and seek an explanation for why it is high. To assess BOExplain's

[3]https://www.kaggle.com/rikdifos/credit-card-approval-prediction
[4]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

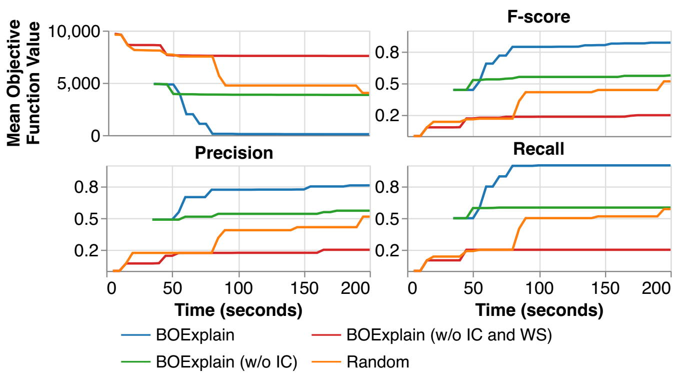


Figure 5: House: best objective function value, F-score, precision, and recall, found at each 5 second increment averaged over 10 runs. The goal is to minimize the objective function. (IC = Individual Contribution Encoding, WS = Warm Start)

efficacy at removing the corrupted tuples, we define the objective function to minimize the distance between the queried result on the passed data and the result of the query issued on the data with the corrupted tuples removed. We use two categorical search variables Neighbourhood and Exterior1st which have 25 and 15 distinct values respectively, and one numerical search variable YearBuilt which has domain [1872, 2010]. The search space size is 7.25×10^6.

In this experiment, we compare three strategies for dealing with categorical variables. The first, BOExplain, is our algorithm with both of the IC encoding and warm-start (WS) optimizations proposed in Section 4. To determine whether encoding categorical values to integers based on IC and using a numerical distribution is effective, we consider a second approach, BOExplain (w/o IC), which uses the warm start optimization from Section 4.2, but uses the TPE categorical distribution to model the variables rather than encoding. The third, BOExplain (w/o IC and WS), is BOExplain without any optimizations.

Each method is run for 200 seconds, and the results are shown in Figure 5. The benefit of the warm start is apparent since BOExplain and BOExplain (w/o IC) outperform the other baselines much sooner. Also, BOExplain significantly outperforms BOExplain (w/o IC) which shows that encoding the categorical values, and using a numerical distribution to model the parameter, leads to BO learning the good region which can optimize the objective function when exploited. The F-score, precision, and recall also demonstrate how BOExplain can significantly outperform the baselines. In this experiment, BOExplain completed on average 274.3 iterations, whereas random completed 1148.4 iterations.

Hyperband Experiment. To evaluate our choice of using TPE, we also compare with a Hyperband implementation. For Hyperband, we use the data sample size as the resource for successive halving. To compare fairly with TPE, we run Hyperband for 200 seconds. We start with a random sample of 12.5% of the data and randomly select predicates to evaluate by the objective function. Next, we select the 50% best performing predicates, and evaluate their quality on a sample size of 25%. This repeats until we evaluate the best predicates on 100% of the data, and finally output the best predicate. The objective function value of the best found predicate averaged over 10 runs is 7666.03, whereas for the TPE-based implementation it is 90.74. Since the goal is to minimize the objective function, TPE performed better. The reason is that TPE with our proposed optimizations for categorical variables prioritized promising predicates early on in the search, whereas Hyperband's exploration-based search strategy could not find good quality predicates as quickly.

5.3.2 **Explanation From Source Data.** In this experiment, we derive an explanation from source data on Credit. We corrupt the source data by setting all applicant records satisfying $-23000 \leq$ `DAYS_BIRTH` $\leq -17000 \wedge 2 \leq$ `CNT_FAM_MEMBERS` ≤ 3 to have a "bad" credit status, affecting 20.1% of the data. Corrupting the data decreases the model accuracy, and we define the objective function to increase the accuracy. We derive an explanation from the source data table `application_record` with the variables DAYS_BIRTH and CNT_FAM_MEMBERS which have domains [-25201, -7489] and [1, 15], respectively, and the size of the search space is 7.06×10^{10}.

The experiment is run for 200 seconds, and the results are shown in Figure 6. On average, BOExplain completes 246.8 iterations and random search completes 319.6 iterations during the 200 seconds. BOExplain significantly outperforms Random at optimizing the objective function, as BOExplain on average attains an objective function value at 51 seconds that is higher than the average value Random attains at 200 seconds. This shows that exploiting promising regions can lead to better explanations, and that BOExplain is effective at deriving explanations from source data that passes through an ML pipeline. Although random search can find an explanation with high precision, BOExplain significantly outperforms Random in terms of F-score.

5.4 Case Studies

To understand how BOExplain performs on real workloads, we present three case studies in this section. These case studies use BOExplain to derive an explanation from three real-world datasets under realistic settings. The derived explanations are insightful, which show BOExplain's effectiveness in real-world applications.

Crowdsourced Mislabels. With the Amazon dataset, the inference accuracy is 93.75%. To investigate whether mislabelled reviews decrease the accuracy, we define the objective function to increase the accuracy. We derive a predicate over the variables Country and TextWordCount (the number of words in the review) in the source dataset. After running BOExplain for 60 seconds, the output predicate is `Country` = "Turkey" $\wedge$ 101 $\leq$ `TextWordCount` $\leq$ 221, which increases the test accuracy to 95%. Upon further inspection, we found that the labelling accuracy of the training data is 93%, but the labelling accuracy of the tuples satisfying the returned predicate is 90%. Hence, BOExplain identified that labellers from Turkey were more likely to mislabel long reviews, which degraded the model.

Bias. With the German credit dataset, the predicted rate of good credit risk for individuals 25 years old or older is 76%, and for individuals under 25 years old it is 57%, hence this dataset is biased. We define the objective function to minimize the predicted rate of good credit risk between those over and under 25 years old, and derive a predicate from the training data with the search variables Purpose and DurationInMonths (duration of the loan).

We ran BOExplain for 60 seconds, and the returned predicate is `Purpose` = "car (new)" $\wedge$ 20 $\leq$ `DurationInMonths` $\leq$ 50. After removing the tuples satisfying this predicate, the predicted rate of having good credit risk is 74% and 68% for people over and under 25, respectively, which is significantly less biased. Moreover, in the training data, the overall good credit risk rate is 71% and 60% for people over and under 25, respectively, however for the tuples satisfying the predicates, the rates are 57% and 17%. Therefore,

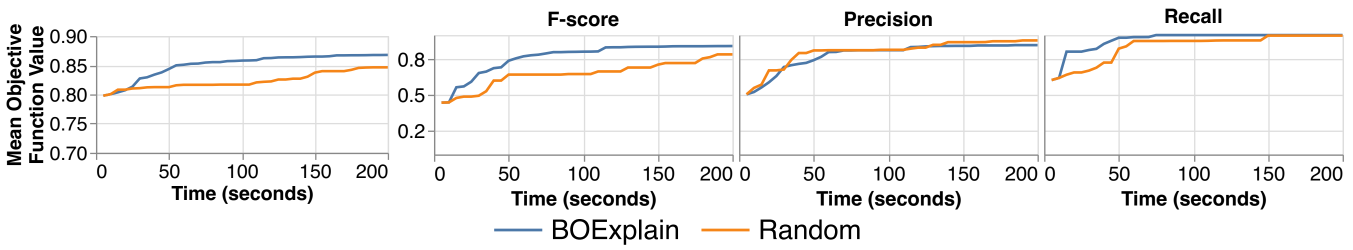


Figure 6: Credit: best objective function value, F-score, precision, and recall found at each 5 second increment, averaged over 10 runs. The goal is to maximize the objective function; larger values are better.

BOExplain identified that long-term loans for new cars are greatly biased in favour of people over 25 years old.

Data Drift. With the NYC dataset, the mean-squared error (MSE) of taxi trip durations is 1801.03 on the training data and 10658.49 on the inference data. The objective function is to minimize the MSE on the inference data, and we search for an explanation over the inference data with the search variables Region and PickUpDateTime.

After running BOExplain for 30 seconds, the returned predicate is `Region = "230"` $\wedge$ `2020-03-12 21:00:00` $\leq$ `PickUpDatetime` $\leq$ `2020-03-31 23:00:00`. When the tuples that satisfy this predicate are removed, the MSE on the inference data is 8706.13. In fact the MSE of the tuples satisfying the predicate is 18151.93, which is much higher than the overall MSE of the inference data (10658.49). Thus, BOExplain identified a region and time period that underwent significant drift as a result of the COVID-19 pandemic.

6 RELATED WORK

Our work is mainly related to query explanation, ML pipeline debugging, and Bayesian optimization.

Query Explanation. BOExplain is most closely related to Scorpion [52] and the work of Roy and Suciu [41]. Both approaches define explanations as predicates. Scorpion uses a space partitioning and merging process to find the predicates, while Roy and Suciu [41] use a data cube approach. Both systems make assumptions about the aggregation query's structure in order to benefit from their white-box optimizations. In contrast, BOExplain supports complex queries, model training, and user defined functions. Variations of these ideas include the DIFF operator [1], explanation-ready databases [40], and counterbalances [34]. Finally, a number of specialized systems focus on explaining specific scenarios, such as streaming data [5], map-reduce jobs [24], online transaction processing workloads [54], cloud services [39], and range-radius queries [46].

Another related concept is the OLAP data cube [21] which is used to explore and discover insights about subsets of multidimensional queries. Much previous work has been dedicated to providing the user with more meaningful and efficient exploration of the data cube [43–45]. Other work has used the *data cube* concept to further understand the results of ML models [12, 13, 37]. However, BOExplain is different from the cube-based approaches in two aspects. First, BOExplain can generate explanations from not only inference data but also training and source data. Second, BOExplain does not need to discretize numerical variables.

ML Pipeline Debugging. Rain [53] is designed to resolve a user's complaint about the result of an inference query by removing a set of tuples that highly influence the query result. In contrast, BOExplain removes sets of tuples satisfying a predicate, which can be easier for a user to understand. In addition, BOExplain is more expressive, and supports UDFs, data science workflows, and pre-processing functions. Data X-Ray [51] focuses on explaining systematic errors in a data generative process. Other systems debug the configuration of a computational pipeline [3, 26, 31, 55].

Optimization Algorithms. Bayesian optimization (BO) is used to optimize expensive black box functions (see [11, 18, 29, 48] for overviews). BO consists of a surrogate model to estimate the expensive, derivative-free objective function, and an acquisition function (e.g., Expected Improvement [47]) to determine the next best point. The most common surrogate models are Gaussian processes [47] (GP) and tree-structured Parzen estimators [8, 10] (TPE). We selected TPE since it scales linearly in the size of the set of evaluated points, whereas a GP scales cubically [10]. Other surrogate models include random forests [23] and neural networks [49].

Hyperband [27] is a bandit-based approach for hyperparameter tuning that uses adaptive resource allocation and early-stopping to speed up random search. We did not choose Hyperband as the optimization approach since a time budget needs to be specified before running the algorithm (whereas TPE can run progressively), and our categorical variable optimizations in Section 4 are designed for a sequential optimization algorithm, which Hyperband is not.

Categorical Bayesian Optimization. Categorical variables in BO are often handled by one-hot encoded [17, 19, 20]. However, this approach does not scale well to variables with many distinct values [42]. BO may use tree-based surrogate models (e.g., random forests [23], TPE [10]) to handle categorical variables, however their predictive accuracy is empirically poor [19, 35]. Other work optimizes a combinatorial search space [6, 15, 36], and categorical and category-specific continuous variables [35]. These works only consider categorical variables or focus on categorical variables with few distinct values, which is unsuitable for query explanation.

7 CONCLUSION

In this paper, we proposed BOExplain, a novel framework for explaining inference queries using BO. This framework treats the inference query along with an ML pipeline as a black-box which enables explanations to be derived from complex pipelines with UDFs. We considered predicates as explanations, and treated the predicate constraints as parameters to be tuned. TPE was used to tune the parameters, and we proposed a novel individual contribution encoding and warm-start heuristic to improve the performance of categorical variables. We performed experiments showing that a) BOExplain can even outperform Scorpion and Macrobase for explaining SQL-only queries in certain situations, b) the proposed IC and warm start techniques were effective, c) BOExplain significantly outperformed random search for explaining inference queries, and d) BOExplain generated useful explanations for real corrupted datasets.

ACKNOWLEDGMENTS

This work was supported in part by Mitacs through an Accelerate Grant, NSERC through a discovery grant and a CRD grant as well as NSF awards 1845638, 2106197, and 1564049, a Columbia SIRS award, Amazon, and Google. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, et al. 2020. DIFF: a relational interface for large-scale data explanation. *The VLDB Journal* (2020), 1–26.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2623–2631.

[3] Cyrille Artho. 2011. Iterative delta debugging. *International Journal on Software Tools for Technology Transfer* 13, 3 (2011), 223–246.

[4] Lucas Baier, Marcel Hofmann, Niklas Kühl, Marisa Mohr, and Gerhard Satzger. 2020. Handling Concept Drifts in Regression Problems–the Error Intersection Approach. *arXiv preprint arXiv:2004.00438* (2020).

[5] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 541–556.

[6] Ricardo Baptista and Matthias Poloczek. 2018. Bayesian optimization of combinatorial structures. *arXiv preprint arXiv:1806.08838* (2018).

[7] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research* 13, 1 (2012), 281–305.

[8] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*. 115–123.

[9] James Bergstra, Dan Yamins, and David D Cox. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, Vol. 13. Citeseer, 20.

[10] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.

[11] Eric Brochu, Vlad M Cora, and Nando De Freitas. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* (2010).

[12] Bee-Chung Chen, Lei Chen, Yi Lin, and Raghu Ramakrishnan. 2005. Prediction cubes. In *Proceedings of the 31st international conference on Very large data bases*. 982–993.

[13] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. 2019. Slice finder: Automated data slicing for model validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1550–1553.

[14] Dean De Cock. 2011. Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project. *Journal of Statistics Education* 19, 3 (2011).

[15] Aryan Deshwal, Syrine Belakaria, and Janardhan Rao Doppa. 2020. Scalable Combinatorial Bayesian Optimization with Tractable Statistical models. *arXiv preprint arXiv:2008.08177* (2020).

[16] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[17] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. In *Automated Machine Learning*. Springer, Cham, 3–33.

[18] Peter I Frazier. 2018. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[19] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. 2020. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing* 380 (2020), 20–35.

[20] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1487–1495.

[21] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery* 1, 1 (1997), 29–53.

[22] Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielcsantos, Hvass-Labs, Mikhail Pak, SoManyUsernamesTaken, Fred Callaway, Loïc Estève, Lilian Besson, Mehdi Cherti, Karlson Pfannschmidt, Fabian Linzberger, Christophe Cauet, Anna Gut, Andreas Mueller, and Alexander Fabisch. 2018. *scikit-optimize/scikit-optimize: v0.5.2*. https://doi.org/10.5281/zenodo.1207017

[23] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 507–523.

[24] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. 2012. Perfxplain: debugging mapreduce job performance. *arXiv preprint arXiv:1203.6400* (2012).

[25] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*. PMLR, 1885–1894.

[26] Rahul Krishna, Md Shahriar Iqbal, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. 2020. CADET: A Systematic Method For Debugging Misconfigurations using Counterfactual Reasoning. *arXiv preprint arXiv:2010.06061* (2020).

[27] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.

[28] Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*. PMLR, 367–377.

[29] Daniel James Lizotte. 2008. *Practical bayesian optimization*. University of Alberta.

[30] Brandon Lockhart, Jinglin Peng, Weiyuan Wu, Jiannan Wang, and Eugene Wu. 2021. Explaining Inference Queries with Bayesian Optimization. https://github.com/sfu-db/BOExplain.

[31] Raoni Lourenço, Juliana Freire, and Dennis Shasha. 2020. Bugdoc: A system for debugging computational pipelines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2733–2736.

[32] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055* (2018).

[33] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. 2014. Causality and Explanations in Databases. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1715–1716. https://doi.org/10.14778/2733004.2733070

[34] Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2019. Going beyond provenance: Explaining query answers with pattern-based counterbalances. In *Proceedings of the 2019 International Conference on Management of Data*. 485–502.

[35] Dang Nguyen, Sunil Gupta, Santu Rana, Alistair Shilton, and Svetha Venkatesh. 2020. Bayesian Optimization for Categorical and Category-Specific Continuous Inputs.. In *AAAI*. 5256–5263.

[36] Changyong Oh, Jakub Tomczak, Efstratios Gavves, and Max Welling. 2019. Combinatorial Bayesian Optimization using the Graph Cartesian Product. In *Advances in Neural Information Processing Systems*. 2914–2924.

[37] Eliana Pastor, Luca de Alfaro, and Elena Baralis. 2021. Looking for Trouble: Analyzing Classifier Behavior via Pattern Divergence. (2021).

[38] Jorge Ramírez, Marcos Baez, Fabio Casati, and Boualem Benatallah. 2019. Crowdsourced datasets to study the generation and impact of text highlighting in classification tasks. (11 2019). https://doi.org/10.6084/m9.figshare.9917162.v4

[39] Sudip Roy, Arnd Christian König, Igor Dvorkin, and Manish Kumar. 2015. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 1167–1178.

[40] Sudeepa Roy, Laurel Orr, and Dan Suciu. 2015. Explaining Query Answers with Explanation-Ready Databases. *Proc. VLDB Endow.* 9, 4 (Dec. 2015), 348–359. https://doi.org/10.14778/2856318.2856329

[41] Sudeepa Roy and Dan Suciu. 2014. A Formal Approach to Finding Explanations for Database Queries. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 1579–1590. https://doi.org/10.1145/2588555.2588578

[42] Binxin Ru, Ahsan Alvi, Vu Nguyen, Michael A Osborne, and Stephen Roberts. 2020. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning*. PMLR, 8276–8285.

[43] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. 1998. Discovery-driven exploration of OLAP data cubes. In *International Conference on Extending Database Technology*. Springer, 168–182.

[44] Sunita Sarawagi and Gayatri Sathe. 2000. i3: intelligent, interactive investigation of olap data cubes. *ACM SIGMOD Record* 29, 2 (2000), 589.

[45] Gayatri Sathe and Sunita Sarawagi. 2001. Intelligent rollups in multidimensional OLAP data. In *VLDB*, Vol. 1. 531–540.

[46] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. 2018. Explaining aggregates for exploratory analytics. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 478–487.

[47] Matthias Schonlau, William J Welch, and Donald R Jones. 1998. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series* (1998), 11–25.

[48] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.

[49] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. 2015. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*. 2171–2180.

[50] Jasper Roland Snoek. 2013. *Bayesian optimization and semiparametric models with applications to assistive technology*. Ph.D. Dissertation. Citeseer.

[51] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. 2015. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1231–1245.

[52] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *Proc. VLDB Endow.* 6, 8 (June 2013), 553–564. https://doi.org/10.14778/2536354.2536356

[53] Weiyuan Wu, Lampros Flokas, Eugene Wu, and Jiannan Wang. 2020. Complaint-driven Training Data Debugging for Query 2.0. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1317–1334.

[54] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 International Conference on Management of Data*. 1599–1614.

[55] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. 2014. Encore: Exploiting system environment and correlation information for misconfiguration detection. In

Proceedings of the 19th international conference on Architectural support for programming languages and operating systems. 687–700.